

2nd Workshop on Fog Computing and the IoT

Fog-IoT 2020, April 21, 2020, Sydney, Australia

Edited by

Anton Cervin

Yang Yang



Editors

Anton Cervin 

Lund University, Sweden
anton@control.lth.se

Yang Yang 

ShanghaiTech University, China
yangyang@shanghaitech.edu.cn

ACM Classification 2012

Computer systems organization → Cloud computing

ISBN 978-3-95977-144-3

Published online and open access by

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at <https://www.dagstuhl.de/dagpub/978-3-95977-144-3>.

Publication date

April, 2020

Bibliographic information published by the Deutsche Nationalbibliothek

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at <https://portal.dnb.de>.

License

This work is licensed under a Creative Commons Attribution 3.0 Unported license (CC-BY 3.0):
<https://creativecommons.org/licenses/by/3.0/legalcode>.



In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/OASlcs.Fog-IoT.2020.0

ISBN 978-3-95977-144-3

ISSN 1868-8969

<https://www.dagstuhl.de/oasics>

OASlcs – OpenAccess Series in Informatics

OASlcs aims at a suitable publication venue to publish peer-reviewed collections of papers emerging from a scientific event. OASlcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

Editorial Board

- Daniel Cremers (TU München, Germany)
- Barbara Hammer (Universität Bielefeld, Germany)
- Marc Langheinrich (Università della Svizzera Italiana – Lugano, Switzerland)
- Dorothea Wagner (*Editor-in-Chief*, Karlsruher Institut für Technologie, Germany)

ISSN 1868-8969

<https://www.dagstuhl.de/oasics>

■ Contents

Preface	
<i>Anton Cervin and Yang Yang</i>	0:vii
Invited Talks	
Fog and Edge Computing: Challenges and Emerging Trends	
<i>Rodrigo N. Calheiros</i>	1:1–1:1
From Vehicular Networks to IoT for Smart Roads: How a Communication Engineer Can Help Solve Transportation Problems	
<i>Guoqiang Mao</i>	2:1–2:1
Regular Papers	
Quality-Of-Control-Aware Scheduling of Communication in TSN-Based Fog Computing Platforms Using Constraint Programming	
<i>Mohammadreza Barzegaran, Bahram Zarrin, and Paul Pop</i>	3:1–3:9
Processing LiDAR Data from a Virtual Logistics Space	
<i>Jaakko Harjuhahto, Anton Debner, and Vesa Hirvisalo</i>	4:1–4:12
Addressing the Node Discovery Problem in Fog Computing	
<i>Vasileios Karagiannis, Nitin Desai, Stefan Schulte, and Sasikumar Punnekkat</i>	5:1–5:10
Routing Using Safe Reinforcement Learning	
<i>Gautham Nayak Seetanadi and Karl-Erik Årzén</i>	6:1–6:8
Real-Time Containers: A Survey	
<i>Václav Struhár, Moris Behnam, Mohammad Ashjaei, and Alessandro V. Papadopoulos</i>	7:1–7:9
Evaluation of Burst Failure Robustness of Control Systems in the Fog	
<i>Nils Vreman and Claudio Mandrioli</i>	8:1–8:8
Next-Generation SDN and Fog Computing: A New Paradigm for SDN-Based Edge Computing	
<i>Eder Ollora Zaballa, David Franco, Marina Aguado, and Michael Stüberr Berger</i>	9:1–9:8
Fog Network Task Scheduling for IoT Applications	
<i>Chongchong Zhang, Fei Shen, Jiong Jin, and Yang Yang</i>	10:1–10:9
Realizing Video Analytic Service in the Fog-Based Infrastructure-Less Environments	
<i>Qiushi Zheng, Jiong Jin, Tiehua Zhang, Longxiang Gao, and Yong Xiang</i>	11:1–11:9
Detection of Fog Network Data Telemetry Using Data Plane Programming	
<i>Zifan Zhou, Eder Ollora Zaballa, Michael Stüberr Berger, and Ying Yan</i>	12:1–12:11



■ Preface

The 2nd Workshop on Fog Computing and the IoT is arranged in conjunction with the CPS-IoT Week in 2020. It is intended as a forum for presenting and discussing recent developments and trends in Fog/Edge Computing that represent challenges and opportunities for CPS and IoT researchers and practitioners. Fog/Edge Computing is a novel and multidisciplinary topic, at the intersection of CPS, IoT and Cloud Computing, and we believe that it benefits from exposure and inputs from CPS and IoT researchers. However, Cloud Computing cannot provide the dependability and quality-of-service guarantees required for industrial applications. Our experience from the “European Training Network on Fog Computing for Robotics and Industrial Automation” (funding 15 PhD students) and the “Nordic University Hub on Industrial IoT” (with over 30 affiliated PhD students) is that the expertise of CPS and IoT researchers is essential to the development of future Fog Computing platforms. This year the workshop received 13 submissions out of which 10 were accepted for oral presentations. The topics of the accepted presentations include fog/edge and IoT aspects of scheduling, system architectures, control analysis, distributed data processing, real-time kernels, node discovery, telemetry, simulation, and machine learning. In addition the workshop has two invited presentations.

Anton Cervin
Yang Yang

March 5, 2020

Short biographies

Anton Cervin is an Associate Professor in Automatic Control at Lund University since 2007. He received the M. S. degree in Computer Science and Engineering and the Ph. D. degree in Automatic Control from Lund University in 1998 and 2003, respectively. Anton Cervin has done research in the intersection between automatic control and computer science for the past twenty years. Developing popular analysis tools such as TrueTime and Jitterbug, he has investigated the interplay between control performance, resource scheduling, and feedback mechanisms in real-time systems. The tools have an extensive user base in industry and academia, the research papers are highly cited, and four of his publications have received best paper awards. He has held a junior researcher grant and three individual project grants from the Swedish Research Council.

Yang Yang is a full professor at ShanghaiTech University, China, serving as the Executive Dean of School of Creativity and Art and the Co-Director of Shanghai Institute of Fog Computing Technology (SHIFT). Before joining ShanghaiTech University, he has held faculty positions at the Chinese University of Hong Kong, Brunel University (UK), University College London (UCL, UK), and SIMIT, CAS (China). His research interests include fog computing networks, service-oriented collaborative intelligence, wireless sensor networks, IoT applications, and advanced testbeds and experiments. He has published more than 200 papers in these research areas. He is a General Co-Chair of the IEEE DSP 2018 conference and a TPC Vice-Chair of the IEEE ICC 2019 conference. Yang is a Fellow of the IEEE.



Fog and Edge Computing: Challenges and Emerging Trends

Rodrigo N. Calheiros 

School of Computer, Data and Mathematical Sciences, Western Sydney University, Australia
R.Calheiros@westernsydney.edu.au

Abstract

Just as the trend of data and computing consolidation via cloud computing starts to fade out, new paradigms that can better handle the unique demand of Internet of Things (IoT) and Big Data emerged in the form of edge and fog computing. Although the literature provides different accounts for the differences between these emerging paradigms, they both rely on computing and storage devices that are closer to IoT devices and users than regular cloud data centers. With the advantage of smaller latencies, they introduce issues such as higher complexity for application development and deployment. In this talk, I will present the context in which these paradigms developed, challenges inhibiting their adoption, and emerging approaches to address some of these issues.

2012 ACM Subject Classification Computer systems organization → Cloud computing

Keywords and phrases Cloud computing, Fog computing, Edge computing, Internet of Things

Digital Object Identifier 10.4230/OASICS.Fog-IoT.2020.1

Category Invited Talk

Funding This work was partially supported by the AWS Cloud Credits for Research program.

Short bio

Dr. Rodrigo N. Calheiros is a Senior Lecturer and Associate Dean, Research in the School of Computer, Data and Mathematical Sciences, Western Sydney University, Australia. He has been conducting research in the area of Cloud computing since 2008, and contributed to diverse aspects in the field including Multiclouds, energy-efficient cloud computing, and Edge computing. He is also one of the original designers and developers of CloudSim, a widely used simulator of Cloud environments. He co-authored more than 70 papers, which attracted together 13,000 Google Scholar citations. His research interests also include Big Data, Internet of Things, Internet Computing, and their application. He is a Fellow of the Higher Education Academy of UK, Senior Member of the IEEE and Senior Member of the ACM.



© Rodrigo N. Calheiros;

licensed under Creative Commons License CC-BY

2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020).

Editors: Anton Cervin and Yang Yang; Article No. 1; pp. 1:1–1:1

OpenAccess Series in Informatics



OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

From Vehicular Networks to IoT for Smart Roads: How a Communication Engineer Can Help Solve Transportation Problems

Guoqiang Mao 

School of Electrical and Data Engineering, University of Technology Sydney, Australia
Guoqiang.Mao@uts.edu.au

Abstract

Intelligent transportation system (ITS) is an important development that applies advanced sensing, communication, big data analysis and control technologies to ground transportation in order to improve safety, mobility and efficiency. This talk will begin with a brief introduction to our work in vehicular networks, which started more than ten years ago. As we delve deeper into vehicular networks and interact more frequently with transportation stakeholders, we realize that ITS is a truly cross-disciplinary area, in order for vehicular networks to achieve its desired impact, we need to think beyond the traditional communication domain, and start to ponder the deeper-level questions of what fundamental changes can be brought by advanced sensing and communication techniques to transportation and how the applications of advanced sensing and communication techniques can help solve crucial transportation problems. To this end, we will introduce our more recent work of developing advanced IoT technology to transform our roads into smart roads, which in the shorter term, make our roads safer and more efficient while providing the fine-grained real-time traffic information for traffic management; in the longer term, provide the much-needed road infrastructure support for the future booming CAV revolution.

2012 ACM Subject Classification Networks → Cloud computing

Keywords and phrases Intelligent transportation systems, Vehicular networks, Internet of Things

Digital Object Identifier 10.4230/OASICS.Fog-IoT.2020.2

Category Invited Talk

Short bio

Guoqiang Mao received a PhD in telecommunications engineering in 2002 from Edith Cowan University, Australia. He was a faculty member at the School of Electrical and Information Engineering, the University of Sydney, between 2002 and 2014. He joined the University of Technology Sydney in February 2014 as Professor of Wireless Networking and Director of Center for Real-time Information Networks. He has published three books and over 200 papers in international conferences and journals, including over 100 papers in IEEE journals, which have been cited over 8,500 times. He is an editor of the IEEE Transactions on Intelligent Transportation Systems, IEEE Transactions on Wireless Communications, IEEE Transactions on Vehicular Technology and received the “Top Editor” award for outstanding contributions to the IEEE Transactions on Vehicular Technology in 2011, 2014 and 2015. He is a co-chair of the IEEE Intelligent Transport Systems Society Technical Committee on Communication Networks. He has served as a chair, co-chair and TPC member in a number of international conferences, and has received best paper awards from several leading international conferences. His research interests include intelligent transportation systems, vehicular networks, Internet of Things, next generation mobile communication systems, and wireless localization techniques. He is a Fellow of the IEEE and IET.



© Guoqiang Mao;
licensed under Creative Commons License CC-BY
2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020).

Editors: Anton Cervin and Yang Yang; Article No. 2; pp. 2:1–2:1

OpenAccess Series in Informatics



OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Quality-Of-Control-Aware Scheduling of Communication in TSN-Based Fog Computing Platforms Using Constraint Programming

Mohammadreza Barzegaran¹ 

DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark
mohba@dtu.dk

Bahram Zarrin 

DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark
baza@dtu.dk

Paul Pop 

DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark
paupo@dtu.dk

Abstract

In this paper we are interested in real-time control applications that are implemented using Fog Computing Platforms consisting of interconnected heterogeneous Fog Nodes (FNs). Similar to previous research and ongoing standardization efforts, we assume that the communication between FNs is achieved via IEEE 802.1 Time Sensitive Networking (TSN). We model the control applications as a set of real-time streams, and we assume that the messages are transmitted using time-sensitive traffic that is scheduled using the Gate Control Lists (GCLs) in TSN. Given a network topology and a set of control applications, we are interested to synthesize the GCLs for messages such that the quality-of-control of applications is maximized and the deadlines of real-time messages are satisfied. We have proposed a Constraint Programming-based solution to this problem, and evaluated it on several test cases.

2012 ACM Subject Classification Networks → Traffic engineering algorithms; Computer systems organization → Embedded software; Theory of computation → Constraint and logic programming

Keywords and phrases TSN, Fog Computing, Constraint Programming, Quality of Control

Digital Object Identifier 10.4230/OASICS.Fog-IoT.2020.3

Funding *Mohammadreza Barzegaran*: The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785, FORA – Fog Computing for Robotics and Industrial Automation.

1 Introduction

In this paper we focus on Fog Computing Platforms (FCPs) for Industrial Control Applications, consisting of heterogeneous *fog nodes* (FNs). We consider that FNs are interconnected using a deterministic communication solutions such as IEEE 802.1 Time Sensitive Networking (TSN) [7]. TSN consists of a set of amendments to the IEEE 802.1 Ethernet standard that introduce real-time and safety critical aspects, e.g., IEEE 802.1Qbv defines a Time-Aware Shaper (TAS) mechanism that enables the scheduling of messages based on a global schedule table. The scheduling relies on a clock synchronization mechanism 802.1ASrev [9], which defines a global notion of time. The configuration of the communication infrastructure in an

¹ corresponding author



FCP has an impact on the performance of controllers, which are the main components of industrial applications. The main focus of this paper is to configure an FCP in terms of the scheduling of messages on TSN [5] such that the control performance is maximized.

Fog Computing has received a lot of attention recently [11], and several researchers have proposed the use of TSN in an FCP as a means of achieving deterministic communication for dependable industrial applications [12]. There has been a lot of work on task scheduling for control performance [14], including considering Fog-based implementations [1]. Although researchers have proposed approaches to derive the schedule tables in TSN for Time-Sensitive (TS) traffic, e.g., via Satisfiability Modulo Theories (SMT) [5] and metaheuristics [13] only one work so far has addressed the issue of control performance [10] for industrial applications. [10] focuses on the problem of routing and scheduling of messages to achieve control stability, but ignores the specifics of scheduling TS traffic in TSN, which does not allow the control of individual frames. Instead, only the status of the queue gates can be controlled via Gate Control Lists (GCLs). This may lead to non-determinism of message scheduling, which has to be carefully considered during the GCL synthesis.

In this paper, we propose a constraint programming (CP)-based GCL synthesis strategy aiming at maximizing the quality-of-control (QoC). We employ meta-heuristic search strategies in CP solvers to reduce the computation time needed to find optimized solutions.

2 System Model

2.1 Architecture Model

We model the system architecture as a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ and a set of routes \mathcal{R} , where \mathcal{V} is a set of vertices that represents nodes, and \mathcal{E} is a set of edges, where an edge represents a physical link between two nodes. A node $\nu_i \in \mathcal{V}$ is either an end-system, which may be the source (talker) or the destination (listener) of a stream, or a switch, which forwards messages to the other nodes. A physical link is a full-duplex bidirectional link $\epsilon_{i,j} \in \mathcal{E}$ (equivalent to $\epsilon_{j,i}$) that logically links the nodes ν_i and ν_j . A logical link $\epsilon_{i,j}$ is characterized by the tuple $\langle s, d, mt \rangle$ denoting the speed of the port in Mbit/s, the transmission delay of the port and the time granularity (macrotick) of an event for the port in micro-seconds. According to IEEE 802.1Qbv [8], we assume 8 queues for each link $\epsilon_{i,j}$ which connects the egress port of the node ν_i to the ingress port of the node ν_j .

The transmission delay of a link, $\epsilon_{i,j}.d$, is captured by the function $\mathbf{h}(c)$ which gets the size of a stream's frame c , as input, and is given for every link. Given that each stream has a known size and it is forwarded through a port with a known speed, the transmission time of the stream's frames can be easily determined. For example, transmitting a maximum transmission unit (MTU)-sized IEEE 802.1Q Ethernet frame of 1,542 bytes on a 1 Gbit/s link would take 12.33 μs . The MTU-sized frame is the maximum size of a single data unit that can be transmitted over a network.

A route $r_i \in \mathcal{R}$ is an ordered list of links, starting with a link originating in a talker end system, and ending with a link in a listener end system. The number of links in the route r_i is denoted with $|r_i|$. We define the function $\mathbf{u} : \mathcal{R} \times \mathbb{N}_0 \rightarrow \mathcal{E}$ to capture the j th link of the route r_i . We assume that each stream is associated to only one route but several streams may share the same route. We also assume that the streams are unicast which impose that there is only one talker and one listener for a stream. Our model can be extended to multicast streams.

2.2 Application Model

We model a control application as a set of streams \mathcal{S} . A stream $s_i \in \mathcal{S}$ is captured by the tuple $\langle p, c, t, d, j \rangle$ denoting the priority, the message size in bytes, the period in milliseconds, the deadline, i.e., the maximum allowed end-to-end delay, and the maximum allowed jitter, both in milliseconds. Since, we assume 8 queues for each link, the priority of a message is given from 0 to 7. The number of instances for stream s_i is denoted with $|s_i|$, and is derived from the period of the stream t and the hyperperiod which is the least common multiple of the periods of all streams. For example, for three streams with the periods of 4, 5 and 3 ms, the hyperperiod would be 60 ms and the streams will have 15, 12 and 20 instances respectively.

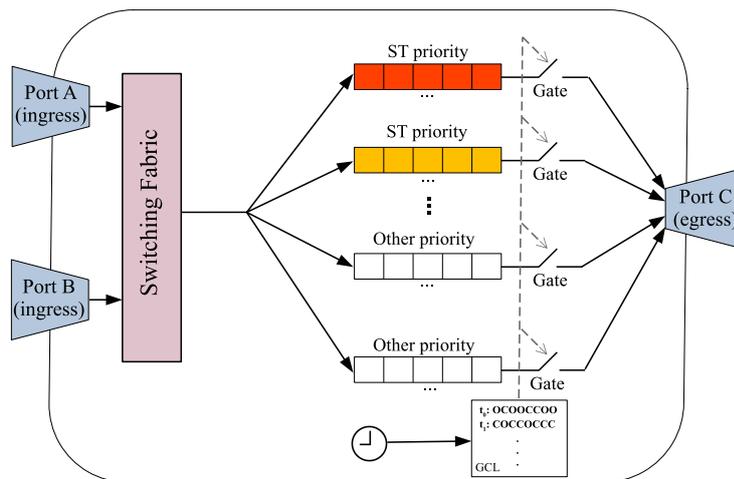
The stream s_i is transmitted via a route r_j which is captured by the function $\mathbf{z} : \mathcal{S} \rightarrow \mathcal{R}$ that maps the streams to the routings. We define a frame for each instance $0 \leq k < |s_i|$ of the stream s_i and on each link $0 \leq m < |r_j|$ of the route r_j , and denote it with $f_{i,m}^k$. A frame $f_{i,m}^k$ is associated with the tuple $\langle \phi, l \rangle$ denoting the start time of the frame (offset ϕ) and its duration (length l).

2.3 Time-Sensitive Transmission in TSN

The internal of a TSN switch is depicted in Fig. 1, where the switching fabric receives streams from ingress ports and forwards each stream to the egress port that is determined in the internal routing tables. In this paper, we assume all the streams are using the Time-Sensitive (TS) class for the transmission.

We assume that each of the egress ports has eight priority queues and each priority queue stores the forwarded stream in First-In-First-Out (FIFO) order. A subset of the queues is reserved for Scheduled Traffic (ST) according to the Priority Code Point (PCP) defined in the frame header; and the remaining queues are used for other, less critical, traffic.

According to the 802.1Qbv standard, a gate is associated to each of the queues which controls the traffic flow by opening and closing that are determined in the predefined Gate Control List (GCL). An open gate only allows the transition of queued traffic from the predetermined egress port. When multiple gates are open at the same time on the same egress port, the highest priority queue blocks other gates until closing.



■ Figure 1 TSN switch internals.

3 Problem Formulation

We formulate the problem as follows: Given (1) a set of streams \mathcal{S} , (2) a network graph \mathcal{G} , and (3) a set of routings \mathcal{R} , we want to determine the GCLs such that the streams are schedulable (their deadlines are satisfied) and the QoC, as defined in Sect. 4, is maximized. In this paper we assume, similar to [5], that the GCLs are deterministic, i.e., the streams are isolated from each other: Only the frames of one of the streams are present in a queue at a time. Hence, the GCL synthesis problem is equivalent to determining (i) the offsets of frames $f_{i,m}^k \cdot \phi$, and (2) their duration $f_{i,m}^k \cdot l$. The offset of a frame maps to when the gate should be open and the duration of the frame maps to how long it should be open.

4 Control Performance

A control application takes input from sensors, processes data, calculates output, and sends the output to actuators. Various communication links are used to link sensors and actuators to the processing elements where control output is calculated. A control application is dependent on time, i.e., timing of data sampling from sensors, calculation of control output and actuation of actuators, which affects the control performance. The control performance is degraded when the delay between sampling and actuation is more than what the controller is designed for or when the delay varies in each iteration, see [3] for more details.

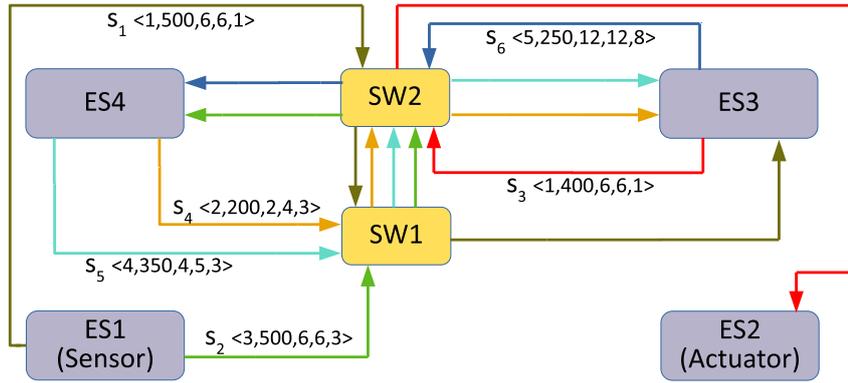
In this paper, we consider that the communication between sensors, processing elements and actuators is based on TSN. We schedule the sensor and actuator messages along with other messages, and the control performance degrades when the control-related messages experience jitter, defined in our case as the variation among the end-to-end delays of a message. We use JitterTime to analyse the Quality-of-Control (QoC), which is used interchangeably to mean “control performance”. JitterTime simulates a control application using the given timing for sampling and actuation and calculates the QoC using the given quadratic cost function, see [4] for details.

For the examples and test cases in the paper, we consider that the control tasks implement a control application consisting of a dynamical system, and we use a quadratic cost function for JitterTime, similar to [1]. The sensor samples the plant with the same period as the control application and sends a message to the node that runs the control application. The actuator receives a message when the control application produces its output. Fig. 2 shows an example system model.

5 Constraint Programming

Constraint Programming (CP) is a declarative programming paradigm that has been widely used to solve a variety of optimization problems such as scheduling, routing, and resource allocations. With CP, a problem is modeled through a set of variables and a set of constraints. Each variable has a finite set of values, called domain, that can be assigned to it. Constraints restrict the variables’ domains by bounding them to a range of values and defining relations between the domains of different variables. The constraint solver systematically performs an exhaustive search by exploring all the possibilities of assigning different values to the variables.

In our future work we will integrate JitterTime with our CP formulation to evaluate *each* visited solution during the search w.r.t. its QoC. However, our approach in this paper is to use the *jitter* as a proxy for the QoC [4]. Hence, we are looking for solutions such that the



■ **Figure 2** Example system model. One control application with messages of 6 ms period on a FCP-based architecture with a sensor, an actuator, two FNs and two switches. The sensor sends a 500 bytes message to ES3 where control output is being calculated and ES3 sends a message of 400 bytes when output is ready. All links have the speed of 100 Mbps. The routing and parameters of streams are also depicted; coloring distinguishes the different streams.

network and stream constraints defined in Sect. 5.2, are satisfied and the jitter defined in Sect. 5.3 are minimized. We record all the solutions visited that have the *best* cost function, and after the search terminates we use JitterTime to determine their QoC.

In the following sections, we present a CP model for our problem, including the decision variables, constraints, and the objective function. Additionally, we propose different search strategies to improve the search speed.

5.1 CP Model

We define decision variables for the offsets and lengths of frames in our CP model, and we bound them in Eq. (1).

$$\forall s_i \in \mathcal{S}, \forall m \in [0, \dots, |s_i|), \forall k \in [0, \dots, |r_j|), r_j = \mathbf{z}(s_i), \epsilon_{v,w} = \mathbf{u}(r_j, k) :$$

$$0 \leq f_{i,m}^k \cdot \phi \leq \frac{s_i \cdot t}{\epsilon_{v,w} \cdot mt} \quad f_{i,m}^k \cdot l = \frac{s_i \cdot c}{\epsilon_{v,w} \cdot s \times \epsilon_{v,w} \cdot mt} \quad (1)$$

5.2 Constraints

We model the network using five constraints that regulate traffic. A directed physical link transmits one frame at a time, i.e., two frames can not share a physical link at any time, which is modeled with the constraint in Eq. (2):

$$\forall s_i, s_j \in \mathcal{S}, i \neq j, \forall m \in [0, \dots, |s_i|), \forall n \in [0, \dots, |s_j|),$$

$$\forall k \in [0, \dots, |r_o|), r_o = \mathbf{z}(s_i), \forall l \in [0, \dots, |r_p|), r_p = \mathbf{z}(s_j), \epsilon_{v,w} = \mathbf{u}(r_o, k), \epsilon_{v,w} = \mathbf{u}(r_p, l) :$$

$$(f_{i,m}^k \cdot \phi + m \times \frac{s_i \cdot t}{\epsilon_{v,w} \cdot mt} \geq f_{j,n}^l \cdot \phi + n \times \frac{s_j \cdot t}{\epsilon_{v,w} \cdot mt} + f_{j,n}^l \cdot l) \vee$$

$$(f_{j,n}^l \cdot \phi + n \times \frac{s_j \cdot t}{\epsilon_{v,w} \cdot mt} \geq f_{i,m}^k \cdot \phi + m \times \frac{s_i \cdot t}{\epsilon_{v,w} \cdot mt} + f_{i,m}^k \cdot l). \quad (2)$$

The constraint in Eq. (3) imposes that a stream propagates from the talker to the listener through the ordered links determined in the mapped routing. It also imposes that the frame can only be scheduled to be transmitted after it has completely received by the node

considering the network propagation delay. According to the 802.1AS clock synchronization mechanism [9], the network precision which is the worst-case difference between the nodes clock in the network, is defined and denoted with δ :

$$\begin{aligned} & \forall s_i \in \mathcal{S}, \forall m \in [0, \dots, |s_i|), \forall k \in [0, \dots, (|r_j| - 1)), \\ & r_j = \mathbf{z}(s_i), \epsilon_{v,w} = \mathbf{u}(r_j, k), \epsilon_{w,x} = \mathbf{u}(r_j, (k + 1)) : \\ & f_{i,m}^{k+1} \cdot \phi \times \epsilon_{w,x} \cdot mt \geq (f_{i,m}^k \cdot \phi + f_{i,m}^k \cdot l) \times \epsilon_{v,w} \cdot mt + \epsilon_{v,w} \cdot d + \delta. \end{aligned} \quad (3)$$

We also isolate streams in different queues of switches to avoid displacement of frames. The constraint in Eq. (4) imposes that either two frames are not received at the ingress port of a switch at the same time or have different priorities, i.e, one frame is received after or before the other one, or has different priority when they are received at the same time, which enforces their order of transmission in the switch schedule, see [5] for more details:

$$\begin{aligned} & \forall s_i, s_j \in \mathcal{S}, i \neq j, \forall m \in [0, \dots, |s_i|), \forall n \in [0, \dots, |s_j|), \\ & \forall k \in [1, \dots, |r_o|), r_o = \mathbf{z}(s_i), \forall l \in [1, \dots, |r_p|), r_p = \mathbf{z}(s_j), \\ & \epsilon_{v,w} = \mathbf{u}(r_o, k), \epsilon_{a,b} = \mathbf{u}(r_p, l), \epsilon_{x,v} = \mathbf{u}(r_o, k - 1), \epsilon_{y,a} = \mathbf{u}(r_p, l - 1) : \\ & ((f_{i,m}^k \cdot \phi \times \epsilon_{v,w} \cdot mt + m \times s_i \cdot t + \delta \leq f_{j,n}^{l-1} \cdot \phi \times \epsilon_{y,a} \cdot mt + n \times s_j \cdot t + \epsilon_{y,a} \cdot d) \vee \\ & (f_{j,n}^l \cdot \phi \times \epsilon_{v,w} \cdot mt + n \times s_j \cdot t + \delta \leq f_{i,m}^{k-1} \cdot \phi \times \epsilon_{x,v} \cdot mt + m \times s_i \cdot t + \epsilon_{x,v} \cdot d)) \vee (s_i \cdot p \neq s_j \cdot p). \end{aligned} \quad (4)$$

The constraint in Eq. (5) imposes that a stream is received by its listener within its deadline, i.e., the time interval between the scheduled transmission of a stream from its talker and the reception of it by the listener is smaller than its deadline:

$$\begin{aligned} & \forall s_i \in \mathcal{S}, \forall m \in [0, \dots, |s_i|), r_j = \mathbf{z}(s_i), \epsilon_{a,b} = \mathbf{u}(r_j, 0), \epsilon_{y,z} = \mathbf{u}(r_j, (|r_j| - 1)) : \\ & f_{i,m}^0 \cdot \phi \times \epsilon_{a,b} \cdot mt + s_i \cdot d \geq \epsilon_{y,z} \cdot mt \times (f_{i,m}^{(|r_j|-1)} \cdot \phi + f_{i,m}^{(|r_j|-1)} \cdot l). \end{aligned} \quad (5)$$

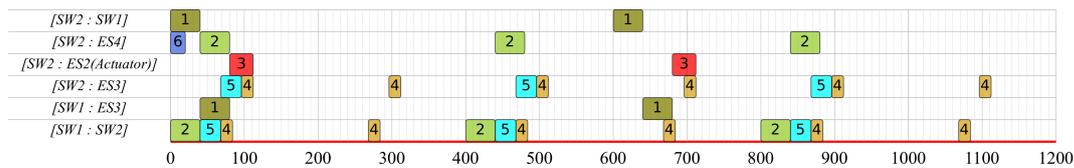
We also define the constraints for the talkers and listeners in Eq. (6), which imposes that the jitter of every instance of a stream should be within the defined value, which is denoted with $s_i \cdot j$ for the stream s_i .

$$\begin{aligned} & \forall s_i \in \mathcal{S}, \forall m, n \in [0, \dots, |s_i|), r_j = \mathbf{z}(s_i), \epsilon_{a,b} = \mathbf{u}(r_j, 0), \epsilon_{y,z} = \mathbf{u}(r_j, (|r_j| - 1)) : \\ & |(f_{i,m}^0 \cdot \phi - f_{i,n}^0 \cdot \phi) \times \epsilon_{a,b} \cdot mt + (m - n) \times s_i \cdot t| \leq s_i \cdot j \\ & |(f_{i,m}^{(|r_j|-1)} \cdot \phi - f_{i,n}^{(|r_j|-1)} \cdot \phi) \times \epsilon_{y,z} \cdot mt + (m - n) \times s_i \cdot t| \leq s_i \cdot j. \end{aligned} \quad (6)$$

5.3 Objective Function

The CP solver finds the first feasible solution that satisfies the presented constraints and determines the values of the CP model variables. The CP solver optimizes the solution concerning the defined objective function. We define an optimization objective to find a solution which schedules streams such that streams have minimum jitter. Although the constraint in Eq. 6 imposes that the jitter is bounded, we seek for a minimum-jitter solution. The proposed optimization objective function Ω accumulates the sending and receiving jitter for every stream, and defined in Eq. (7):

$$\begin{aligned} & \forall s_i \in \mathcal{S}, \forall m, n \in [0, \dots, |s_i|), r_j = \mathbf{z}(s_i), \epsilon_{a,b} = \mathbf{u}(r_j, 0), \epsilon_{y,z} = \mathbf{u}(r_j, (|r_j| - 1)) : \\ & \Omega = \sum |(f_{i,m}^0 \cdot \phi - f_{i,n}^0 \cdot \phi) \times \epsilon_{a,b} \cdot mt + (m - n) \times s_i \cdot t| \\ & + |(f_{i,m}^{(|r_j|-1)} \cdot \phi - f_{i,n}^{(|r_j|-1)} \cdot \phi) \times \epsilon_{y,z} \cdot mt + (m - n) \times s_i \cdot t| \end{aligned} \quad (7)$$



■ **Figure 3** Optimized GCL for the system in Fig. 2. Messages have the same color as the streams in Fig. 2.

5.4 Search Strategies

In this work, we use Google OR-Tools [6] as a CP solver to implement the presented CP model. This CP solver is quite flexible and comes with several extension mechanisms that allow customizing and combining different search strategies such as systematic search, local search, and meta-heuristics algorithms. In this paper, we have used two search strategies; a *Systematic*, and a *Meta-heuristic* strategy.

The first strategy finds the optimal solution by systematically exploring all the possibility of assigning different values to the decision variable. It requires to specify two procedures for the search algorithm. The first is the order of selecting the variables for assignment. The other procedure is the order of selecting the values from the domain of a variable for assignment. Based on our parameter tuning experiments, we choose to use the random order for both procedures (random-variable and random-value).

The second search strategy does not guarantee optimality. Instead, it aims at finding good quality solutions in a reasonable time, and hence it is based on Tabu Search meta-heuristic algorithm [2], which aims to avoid the search process being trapped in a local optimum by increasing diversification and intensification of the search. We have implemented this strategy by extending the OR Tools' implementation of Tabu Search. For intensification, it will keep certain variables bounded to certain values, and for diversification, we will forbid some variables to take some values. We specify two sets of variables for *keep-tenure* and *forbid-tenure*. The variables in the first set must keep their values in the next solution, while the variables in the second set can not use the corresponding values. We also specify the number of iterations or a certain amount of time to keep these variables in these sets.

We run these search strategies for solving the offset variables $f_{i,m}^k \cdot \phi$ as the primary decision variables since they have a direct impact on our cost function. We solve the length variables $f_{i,m}^k \cdot l$ as a constraint satisfaction problem by using *SolveOnce* strategy of the solver which finds the first feasible assignments for these variables.

6 Evaluation

We have evaluated our proposed CP model with several test cases. Our solution is implemented in Java using Google OR-Tools [6] and was run on a computer with an i9 CPU at 3.6 Ghz and 32 GB of RAM, with a time limit of 30 minutes to 5 hours, depending on the size of the test case.

Let us consider the test case in Fig. 2. We schedule the traffic using the *Systematic* and *Meta-heuristic* strategies from Sect. 5.4. Both strategies found the same best solution depicted in Fig. 3 as Gantt chart, which in this case has zero jitter and all streams are schedulable. JitterTime reports a QoC value of 1214 for both solutions. We also measured the run-times of each search strategy, which are 3.67 s for the *Systematic* strategy and 162 ms for the *Meta-heuristic* strategy.

We have also evaluated our solution on progressively larger test cases. The results are presented in Table 1, which shows the 5 additional test cases; Test case 5 is a realistic automotive test case which consists of a TSN-based “fog nodes on wheels” implementation of autonomous driving functions. In the table, the topology of the network is summarized in columns 3 and 4, where we have the number of end-systems and switches, respectively. The values in the column 5 are the maximum jitter for all streams. The values in columns 6 and 7 are the run-time of the solution for respectively *Systematic* and *Meta-heuristic* strategies. As we can see, our CP-based approach is able to find schedulable solutions with zero jitter in all cases. In addition, the *Meta-heuristic* solution scales well with the problem size, and has been able to find the same the optimal solutions as the ones found by the *Systematic* search, in a much shorter time.

However, the improvement in run-time depends on the test case: the search strategy has a big impact on run-time of the solver and the proposed *Meta-heuristic* strategy improves run-time of the addressed scheduling problem.

■ **Table 1** Evaluation results for five test cases.

#	No. of Streams	No. of ESs	No. of SWs	Max. Jitter	Run-Time for Systematic	Run-time for Meta-heuristic
1	10	5	5	0	14:12 min	15.89 s
2	8	5	2	0	2:23 min	3.59 s
3	20	15	15	0	24:44 min	32.2 s
4	20	15	15	0	26:56 min	39.9 s
5	27	20	20	0	42:43 min	2:41 min

7 Conclusions and Future Work

In this paper, we have addressed the problem of real-time communication scheduling on TSN networks on an FCP, aiming at improving the control performance. We have used the scheduled traffic class, which sends the messages based on Gate Control Lists. We have proposed a constraint programming-based solution, modeling the problem constraints as well as objective function for optimizing the network for control applications. The search uses jitter as a “proxy” objective function for the control performance, which has been determined using JitterTime for the best solutions found by the Google OR-Tools solver. As the results show, employing a metaheuristic search in the solver, we can obtain good quality solutions in a short time.

In our future work, we plan to (i) integrate JitterTime into the search process of the CP solver, (ii) integrate task scheduling and message scheduling into a joint QoC-aware CP formulation, and (iii) evaluate the CP approach on larger test cases.

References

- 1 M. Barzegran, A. Cervin, and P. Pop. Towards quality-of-control-aware scheduling of industrial applications on fog computing platforms. In *Workshop on Fog Computing and the IoT*, 2019.
- 2 Edmund K Burke, Graham Kendall, et al. *Search methodologies*. Springer, 2005.
- 3 A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén. How does control timing affect performance? analysis and simulation of timing using Jitterbug and TrueTime. *IEEE Control Systems Magazine*, 23(3):16–30, June 2003.

- 4 A. Cervin, P. Pazzaglia, M. Barzegaran, and R. Mahfouzi. Using JitterTime to analyze transient performance in adaptive and reconfigurable control systems. In *IEEE International Conference on Emerging Technologies and Factory Automation*, pages 1025–1032, 2019.
- 5 Silviu S Craciunas, Ramon Serna Oliver, Martin Chmelfk, and Wilfried Steiner. Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pages 183–192, 2016.
- 6 Google. Google OR-Tools. <https://developers.google.com/optimization>, Accessed on Jan 2020.
- 7 IEEE. *Official Website of the 802.1 Time-Sensitive Networking Task Group*, 2016 (accessed December. 12, 2018). URL: <http://www.ieee802.org/1/pages/tsn.html>.
- 8 IEEE. 802.1Qbv—enhancements for scheduled traffic. <https://www.ieee802.org/1/pages/802.1bv.html>, 2016 Draft 3.1.
- 9 IEEE. 802.1ASrev—timing and synchronization for time-sensitive applications. <http://www.ieee802.org/1/pages/802.1AS-rev.html>, 2017.
- 10 Rouhollah Mahfouzi, Amir Aminifar, Soheil Samii, Ahmed Rezine, Petru Eles, and Zebo Peng. Stability-aware integrated routing and scheduling for control applications in Ethernet networks. In *Design, Automation & Test in Europe Conference*, pages 682–687, 2018.
- 11 Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H. Glitho, Monique J. Morrow, and Paul A. Polakos. A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges. *IEEE Communications Surveys and Tutorials*, 20(1):416–464, 2018.
- 12 P. Pop, M. L. Raagaard, M. Gutierrez, and W. Steiner. Enabling fog computing for industrial automation through Time-Sensitive Networking (TSN). *IEEE Communications Standards Magazine*, 2(2):55–61, 2018.
- 13 Paul Pop, Michael Lander Raagaard, Silviu S Craciunas, and Wilfried Steiner. Design optimisation of cyber-physical distributed systems using IEEE Time-Sensitive Networks. *IET Cyber-Physical Systems: Theory & Applications*, 1(1):86–94, 2016.
- 14 Zhi Wen Wang and Hong Tao Sun. Control and scheduling co-design of networked control system: Overview and directions. In *in Proceedings International Conference on Machine Learning and Cybernetics*, volume 3, pages 816–824, 2012.

Processing LiDAR Data from a Virtual Logistics Space

Jaakko Harjuhahto

Aalto University, Department of Computer Science, Espoo, Finland
jaakko.harjuhahto@aalto.fi

Anton Debner

Aalto University, Department of Computer Science, Espoo, Finland
anton.debner@aalto.fi

Vesa Hirvisalo

Aalto University, Department of Computer Science, Espoo, Finland
vesa.hirvisalo@aalto.fi

Abstract

We study computing solutions that can be used close to the network edge in I2oT systems (Industrial Internet of Things). As a specific use case, we consider a factory warehouse with AGVs (Automated Guided Vehicles). The computing services for such systems should be dependable, yield high performance, and have low latency. For understanding such systems, we have constructed a hybrid system that consists of a simulator yielding virtual LiDAR sensor data streams in real-time and a sensor data processor on a real cluster that acts as a fog computing node close to the warehouse. The processing merges the observations done from the individual sensor streams without using the vehicle-to-vehicle communication links for the complicated computing. We present our experimental results, which show the feasibility of the computing solution.

2012 ACM Subject Classification Computer systems organization → Embedded and cyber-physical systems; Computing methodologies → Modeling and simulation; Computing methodologies → Distributed computing methodologies

Keywords and phrases simulation, hybrid systems, new control applications, fog computing

Digital Object Identifier 10.4230/OASICS.Fog-IoT.2020.4

Funding This work has been financially supported by the Technology Industries of Finland Centennial Foundation.

Acknowledgements We would like to thank research assistant Matias Hyypä for his work on the dataset creation tools and the anonymous reviewers of this paper for their valuable comments.

1 Introduction

In this paper, we address the computing structures that are needed in intelligent traffic systems for warehouse logistics and in the related research and development work. The traditional systems rely on central controllers that coordinate the motion of the vehicles. Recent developments in AI (Artificial Intelligence) are enabling many new approaches including autonomous driving that relies heavily on rich sensor data collected on the traffic situations.

The systems need to process large amounts of sensor data in real-time to maintain an understanding of the ever-changing traffic situation. The computing services for such systems should be dependable, yield high performance, and have low latency. The traditional terminal computing devices (e.g., on the vehicles) or cloud computing services based large data centers are not sufficient. Fog computing solutions are one option to enable such applications (see [10] and [20] for related approaches).



© Jaakko Harjuhahto, Anton Debner, and Vesa Hirvisalo;
licensed under Creative Commons License CC-BY
2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020).

Editors: Anton Cervin and Yang Yang; Article No. 4; pp. 4:1–4:12

OpenAccess Series in Informatics



OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The fog computing paradigm addresses the ways of acting between devices and centralized cloud services, utilizing resources in between them, and thus allowing sufficient resources close to the devices. Even though some standards exist (e.g., [7]), many aspects of the problem of communicating and managing resources on the Cloud to Things continuum need more research. There has been several proposals and studies on the clusters (also mini data centers, cloudlets, etc.) needed in fog computing solutions [22].

Warehouse AGVs are rather typical mobile robots. Their operation is complex including localization, motion planning, and control. Traditionally their warehouse environment is augmented to ease their operation (by using markers, reflectors, etc.). As flexibility is essential and human presence is often needed, the technology is developing toward natural navigation. Such navigation solutions are often based on sensors perceiving the environment, and the development of such systems typically calls for suitable simulators [4].

Our study addresses LiDAR (laser scanner) data processing for AGV coordination. For efficient coordination of the flow of the traffic inside a warehouse, the LiDAR data of the participating vehicles is needed. Using the shared data, vehicles can also help each other to see around corners, and thus, avoid being over-cautious, when there is human presence in a warehouse. However, constructing a shared real-time view calls for plenty of communication. Using, e.g., V2V (Vehicle-to-Vehicle) links for computing the shared view can cause massive use of the wireless communication links.

Our contribution consists of three parts. Firstly, we have built a hybrid setup for research and development purposes. Our hybrid setup uses a virtual warehouse with virtual AGVs and a real cluster for their data processing. Secondly, we have implemented LiDAR data processing that is suitable for control algorithms and does the computing of the shared view within the cluster. Our approach supports both scalability and fault-tolerance of the processing. Thirdly, we present performance measurements that show the feasibility of our approach.

The structure of this paper is as follows. We begin by reviewing AGV systems for warehouse logistics and describing our warehouse case with the simulation model that we have made in Section 2. We continue by explaining the designed computation and communication architecture in Section 3. We describe our hybrid simulation setup and our LiDAR data processing in Section 4. We present our experimentation with the setup in Section 5, and discuss our results in Section 6. We end the presentation with our conclusions.

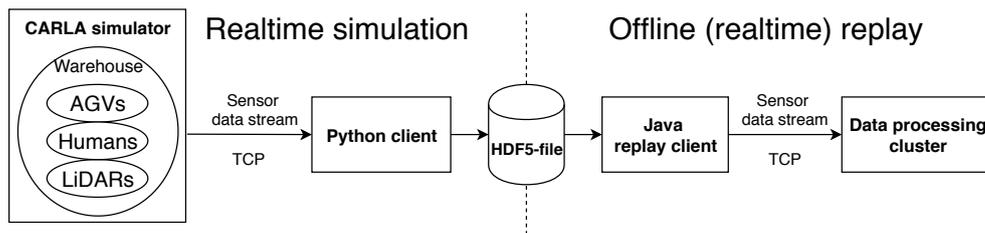
2 AGVs in a Factory Warehouse

We have made a model of a warehouse hall. Our modeling is motivated by the modern factory warehouses. In this section, we first review modern factory warehouses in Subsection 2.1, and then, describe our modeling in Subsection 2.2.

2.1 Modern Factory Warehouses

The operation of manufacturing plants depend on logistics. Manufacturing plants typically have warehouse spaces, through which the goods needed in production flow. The operation of warehouses calls for careful optimization as everything needed should be available, but storing excessive amounts of goods should be avoided as the storage costs are usually high. In addition to speed, flexibility is essential as factories typically have to adjust their operation frequently.

Warehouse operation in factories is still mostly manual, but automation is entering the scene. Warehousing in factories of the future can rely on AGVs and integrated systems for logistics. AGVs typically move goods between locations in a plant environment and



■ **Figure 1** Overview of our setup. CARLA is first used to run our warehouse simulation and produce sensor data streams in real-time. These streams are captured with a Python client and stored in a hierarchical data format (HDF5). The HDF5 files can then be used to replay the real-time sensor data streams without running the CARLA simulator. The advantage of this approach is full reproducibility and control over sending data to our data processing cluster.

the warehouse is central for them. The AGV system is usually controlled by a centralized Warehouse Management System (WMS). Both the navigation of the AGVs in flexible warehouse environments and their co-operation with humans call for advanced sensing technology.

Sabattini et al. [19] give a view on advanced sensing technology for AGV systems and the related warehouse operations. Currently, LiDARs are the typical main sensor for AGVs as they directly yield distance data. However, obstacles limit the view of LiDARs, and limited understanding of a traffic situation can cause unnecessary slow-downs or complete stops for the AGVs. This underlines the need for shared sensing and sensor fusion.

In mobile robotics, understanding of traffic situations is often done by using two-dimensional occupancy grids [4]. An occupancy grid is an abstract representation of the physical situation, where each grid cell indicates the state of the corresponding physical place. Occupancy grids can be used together with robot control algorithms (see, e.g., [14]). The predictions of movements are important for such algorithms, and the history of observations is useful for such predictions [15]. The coordination of multiple robots in intersections presents an important and challenging optimization problem, for which DMPC (Distributed Model Predictive Control) methods are promising [11]. Our design of data processing has been impacted by the needs of such algorithms.

2.2 A Model for a Warehouse

Our goal was to create a simple, easily modifiable virtual warehouse. This was achieved by creating a grid-based structure from modular squares and storage shelf units. Each square is $5 \times 5 \text{ m}^2$ in size, leaving a moderately large working space between the storage shelves. A portion of the resulting warehouse is shown in Figure 2. The modular nature of the warehouse enabled us to experiment with various sizes, for example, varying the storage area from $20 \times 20 \text{ m}^2$ to $50 \times 50 \text{ m}^2$. As the AGVs sense their surroundings only through LiDARs, the graphical details of the warehouse are not important. While the shelf-models appear to be empty in Figure 2, we simplify the scenario by assuming that they are fully populated by stored objects and therefore preventing LiDARs from seeing through the shelves at all.

3 Computing and Communication Architecture

We consider an AGV system that uses LiDAR sensing for shared environment perception. As walls limit the sensing, the halls of the whole plant form distinct physical areas, where shared sensing is the most useful. Thus, in our design we concentrate on sensing inside a

4:4 Processing LiDAR Data from a Virtual Logistics Space

single hall and assume that the processing of the LiDAR data from the hall is done by a single fog node (i.e., a computing cluster). A large plant can have multiple fog nodes, each serving one or more distinct areas.

Our architecture design is inspired by the work of Farkas et al. [5] in many ways. They describe a rather generic approach for using 5G-TSN systems (5G integrated Time-Sensitive Networking) for industrial applications. The integration of 5G and TSN is rather complex, but there exists extensive documentation for both 5G systems and TSN systems (see, e.g., [17] for further information). However, from the communication perspective of an application much of the complexity of a 5G system can be abstracted by a TSN system on top of it.

The architecture as such does not limit the number of the AGVs connected to a fog node or the number of compute nodes inside the cluster. However, the computation and communication capacity of a fog node limit these numbers in practice.

Figure 3 describes our design. The AGVs are connected to the fog node using TSN connections over a wireless 5G network. The whole system runs under central control (SDN controller), which includes the related CUC (Centralized User Configuration) and CNC (Centralized Network Configuration) elements. The controller coordinates both the distributed mini-datacenter (i.e., the fog nodes) and the integrated 5G-TSN system. We assume the cluster intra-connections to be much faster than the wireless connections through separate interfacing (IF) toward the AGVs.

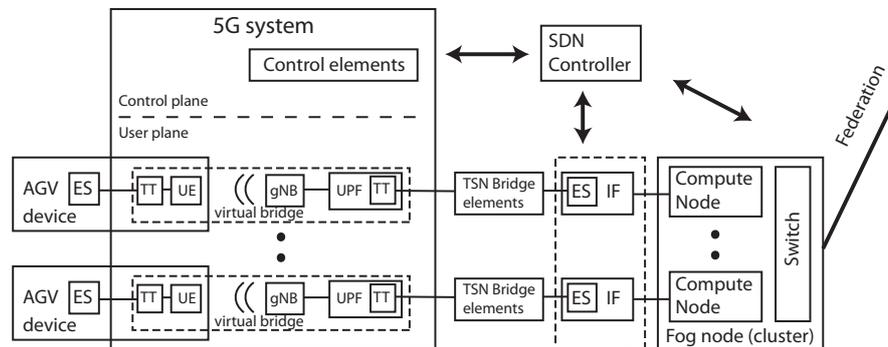
The connection in TSN system exists between the TSN end stations (ES). The connections appear as TSN bridges that are virtualized on top of the underlying 5G system. The 5G system has TSN translation (TT) functionality for mapping the user and control planes towards TSN. This mapping is essential in hiding the 5G details from the TSN connections. In our current design, we use only single PDU (Protocol Data Unit) sessions between the end points. Inside the 5G system, User Plane Functions (UPF) connect to the AGVs through the links between the base stations (gNB). From the 5G system viewpoint, the AGVs appear as UEs (User Equipment). In our current design, we have only one UE within an AGV.

4 Hybrid Processing Setup

Our hybrid setup is based on the CARLA simulator [3] producing virtual sensor data streams and a real cluster processing these data streams. The setup is illustrated in Figure 1. The sensor streams produced by the simulator are stored in a dataset file. The details of the simulation and virtual data streams are described in Subsection 4.1.



■ **Figure 2** Virtual model of a warehouse, where workers can walk freely among autonomous vehicles.

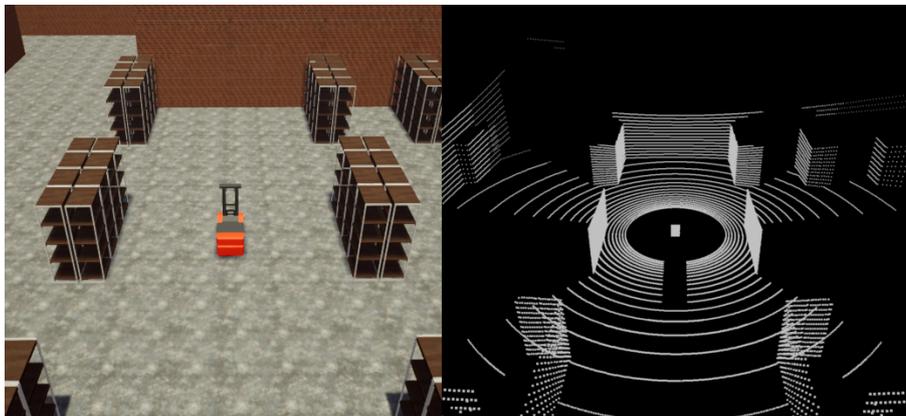


■ **Figure 3** The designed architecture for a single fog node and its connections in the system. The fog nodes can be federated under a single SDN (Software Defined Networking) controller. The AGV devices (left) implement both the TSN (Time Sensitive Networking) end stations and act as 5G system UEs. The fog node (right) communicates with the AGVs by using TSN connections over the wireless 5G network.

The sensor data streams are replayed from the stored dataset file in real-time and processed by the cluster. In our setup, the replay clients simulate the 5G-TSN communication. The modeling and simulation of the 5G-TSN communication is described in Subsection 4.2.

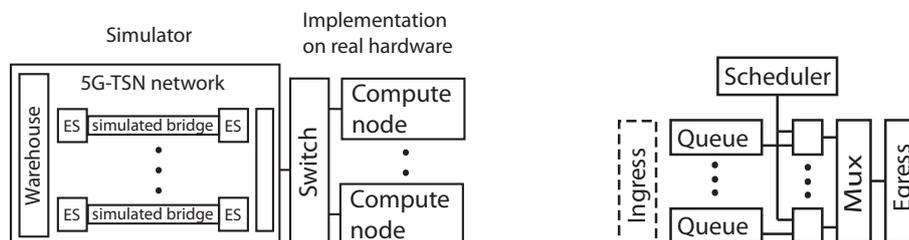
The processing cluster implements state sharing by distributing the observations computed from the LiDAR data streams. The details of the processing cluster and the related experiment are described in Subsection 4.3.

4.1 Generation of Virtual Sensor Data



■ **Figure 4** An example of the data produced by CARLA with a LiDAR sensor attached to an AGV. Left: RGB view of the simulation scene. Right: 3D view of the LiDAR data.

As shown on the left side of Figure 1, we used CARLA to simulate AGVs and humans moving together inside a warehouse. CARLA [3] is an open-source simulator made for autonomous driving research. Its main features include modern rendering pipeline, animated pedestrians, fully controllable vehicles, pre-made urban cities and various simulated sensors, such as cameras and LiDARs. Python clients can be used to control the simulation actors and process sensor data remotely over TCP.



■ **Figure 5** On the left side, the simulation of the 5G-TSN system within the hybrid set-up is shown. The replay client acts as a simulator that communicates with the cluster. On the right side, details of the TSN connection simulation are shown.

We created a Python client for collecting data from the simulation and storing it in to a dataset with hierarchical data format (HDF5 [21]). The dataset used in our experiment is available at [2]. For the work described in this paper, the relevant stored data are LiDAR point clouds, simulation actor positions and rotations for every time step. Velocities of the actors and data from all other kinds of sensors can also be included in a dataset on demand, enabling further development branches for more advanced logistic experiments.

CARLA produces the LiDAR data by performing raycasts from the rotating LiDAR sensor on each simulation step. Each raycast returns the point of first collision with any other object along the ray. The resulting data is essentially a set of coordinates in a 3D space, where the origo is the sensor itself. An example of this data is visualized in Figure 4. While the LiDAR sensor attempts to imitate its real-life counterparts, it is not completely realistic in the sense that the measurement are absolute ground truths without any noise or reflections. Such artefacts can be added to simulate realistic conditions.

4.2 Real-time Simulation

The stored datasets represent situations, where AGVs move and perceive their environment. As illustrated in Figure 1, these situations are replayed from HDF5 files in real-time. From the view point of the application, this is identical to a situation, where the CARLA simulator would be directly connected to the fog system.

The fog system is modeled within the replay client, which acts as a real-time simulator. The setup is illustrated in Figure 5. The communication between warehouse simulator (i.e., a replay based on a CARLA data set) and the fog cluster consists of real data items, but their motion in the larger communication system is simulated. Real data transmissions happen between the fog simulator and the computing cluster as the computing cluster is not virtual but consists of real pieces of hardware. Also, the data transmission within the cluster are real.

In the simulation of the fog system, we do not simulate the underlying 5G system in detail. Instead of the detailed simulation, we simulate the TSN bridges on top of the 5G system. In our setup, their main function is the wireless communication within the AGV system.

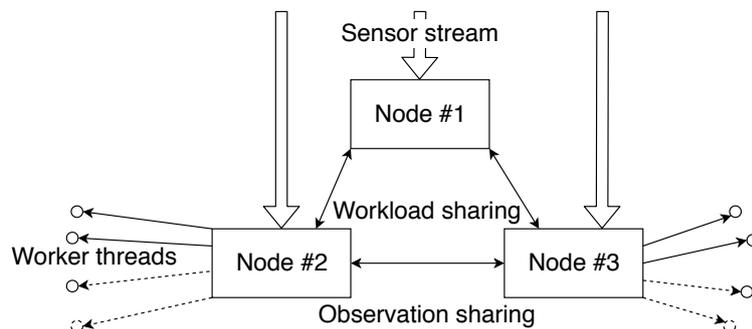
Simulating the operation of the TSN connections is illustrated in the Figure 5 (right side). There can be multiple streams that go over a TSN connection, but we do not model any hierarchy between the streams. Further, there is no modeling of any complex underlying

functions (e.g., traffic shaping). The entering streams (Ingress) are buffered into queues that are scheduled into a time division multiplexer (Mux) before being sent (Egress). Thus, the simulation model is rather abstract compared to a real TSN system on top of a 5G network, but it allows for the testing of various real-time scheduling algorithms together with realistic delay models of the processing and communication steps.

4.3 Intelligent Traffic Coordination with a Cluster

We use a cluster of small compute nodes to maintain the state of the occupancy grid and process LiDAR point clouds from AGVs. The server software is written in C++ and communicates with AGVs and peer nodes with TCP sockets. Data is serialized using FlatBuffers [6]. Every node runs the same software and maintains a copy of the occupancy grid state to provide redundancy and availability. In case of node failures, AGVs using the cluster may simply switch to another node.

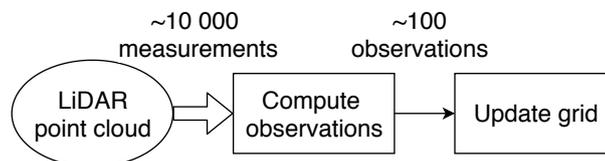
Each cluster node can receive data from AGVs. Once a node receives a LiDAR point cloud, it will check if it has local capacity to process the data frame. If a local worker thread has signaled that it is ready to pull work, the frame is dispatched to that worker. If no local worker capacity is available, the data frame is forwarded to the peer node with the most capacity available at the moment. Each node reports on its available worker capacity to the rest of the cluster. Figure 6 shows the related communication patterns. Once the cluster has accepted a LiDAR data frame from an AGV, it guarantees processing of that frame, barring hardware failure.



■ **Figure 6** Processing cluster. The cluster consists of computation nodes, that can share their workload and observations between each other. Each node has a pool of worker threads. The number and capabilities of these threads depend on the node’s hardware resources. Each node is capable of receiving sensor (LiDAR) data from the AGVs. While the ability to share resources enable flexible setups, in this image each node is receiving a sensor stream from a single AGV.

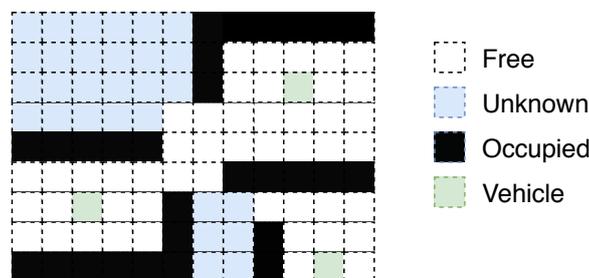
Processing a set of LiDAR data points is a two-phase process, as shown in Figure 7. During the first step, we compute which cells the LiDAR rays either hit or pass through adapting Bresenham’s line algorithm [1] to our occupancy grid. We consider LiDAR ray hits as having priority. If LiDAR rays have both hit something in a cell and passed through the cell, we consider the cell *occupied*. These *observations* on the states of a subset of all cells in the occupancy grid are collected and published to every peer node in the cluster. During the second step, replicated on each node, the observations are committed to the grid data structure. Finally, the node that received the LiDAR data from an AGV will calculate the heuristic cell state and return the entire grid to the AGV. This updated occupancy grid also includes all the observations from other AGVs applied to the compute nodes grid. The

occupancy grid supports concurrent updates from multiple LiDAR frames by implementing concurrency controls on the level of individual cells. If multiple updates overlap, all the observations are recorded to be used as inputs for determining the cell's current state later on.



■ **Figure 7** Simplified process view. In our case, each set of LiDAR measurements consists of a point cloud with over 10 000 points in a 3D coordinate system. These measurements are then squeezed into a far fewer number of observations. Each observation determines the state of a single cell in the grid at that point in time. These observations are then combined with the previous information of the grid, in order to form an updated understanding of the environment.

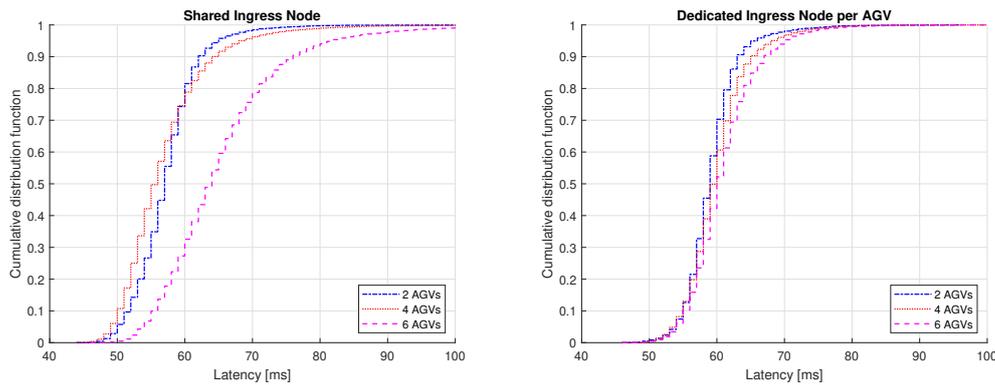
Our model of an occupancy grid is static and regular. Using a static grid of predetermined resolution allows all of the compute nodes to use the same coordinates for occupancy grid cells without necessitating a fully consistent state coherency protocol stemming from the use of dynamic grids. For each cell in the grid, we store the timestamp of the most recent observations of each state we track: 'free', 'occupied' and 'vehicle'. Cells that have never been observed are considered as 'unknown'. The category 'vehicle' cells are derived directly from the positions reported by each of the AGVs. We apply an exponential decay term ($P(t) = e^{-\gamma t}$) to model diminishing trust in the cell's state as time progresses, unless new observations are made, refreshing the timestamps. Suitably chosen constants γ for each state category allows the AGVs using the occupancy grid for guidance decisions to consider the reliability of the knowledge on the current state of individual cells. Cells never explicitly revert back to an 'unknown' state, but AGVs should consider cells with a low reliability as effectively unknown. Figure 8 presents an example of a small occupancy grid.



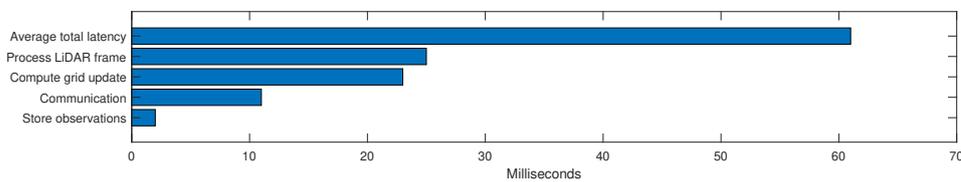
■ **Figure 8** Distributed occupancy grid. Multiple AGVs collaborate to create a shared understanding of the surrounding environment. The grid is divided into cells, that represent the latest observations made from the AGV sensor data.

5 Experimental Results

We validated our design by performing an experiment by using pre-recorded sensor data as described in Section 4.1 to stream LiDAR point clouds to the cluster. The cluster hardware is Intel Atom x5-8350 based commodity-off-the-shelf (COTS) computers connected to a router. The cluster consists of seven compute nodes, all running Ubuntu 18.04 LTS. An additional workstation computer was used to read the sensor data from HDF5 files and push data frames to the cluster at regular 100 ms intervals. We used a warehouse model of 50 meters by 50 meters and an occupancy grid of 1 m by 1 m cells.



■ **Figure 9** Cumulative distribution of end-to-end latencies in milliseconds. On the left, a single compute node acts as the ingress point for all of the LiDAR data produced by AGVs. On the right, each AGV connects to a distinct compute node.

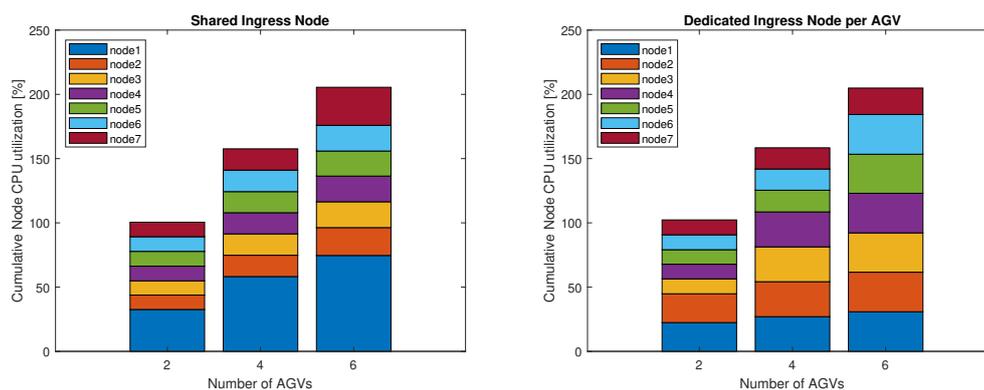


■ **Figure 10** Breakdown of how the individual steps in the end-to-end process, described above, contribute to the observed average total latency. The values are from averaging the results over all of the tests.

The simulation consists of one human walking across the entire warehouse, while 2 to 6 robots follow their own predetermined paths between the storage shelves. The data is collected over a 30 second simulation at 10 samples per second, which matches the 100 ms replay interval. The LiDARs are rotating around their axis at 10 Hz, which means that each LiDAR sensor produces a full 360 degree scan of the environment during each sample. Each LiDAR produced 2500 data points per frame, or 25000 points per second. The dataset is designed to start and stop in roughly the same state configuration to support looped replay.

In the experiment, we measure the end-to-end latency from the point where LiDAR data frame passes through the simulated 5G bridge to the cluster to the point in time when the cluster has returned a new version of the entire occupancy grid over the same simulated 5G connection. Our simulated network offered 500 Mbit/s of upstream and downstream bandwidth divided fairly across every active connection. We perform this latency measurement for 2, 4 and 6 simultaneous AGVs using two strategies for transferring data frames to and from the cluster. In the first arrangement, a single compute node in the cluster acts as service endpoint for all of the AGVs, accepting LiDAR frames and routing these to peer nodes for processing. In the second arrangement, every AGV connects directly to a distinct compute node so that the nodes have sufficient local capacity to process the LiDAR frame and share only the computed observations with the rest of the cluster. Data is collected over 6k LiDAR frames per AGV. Figure 9 presents the cumulative distribution functions of these latency measurements. Additionally, Figure 10 presents a breakdown of the relative contribution to the total latency by each of the phases in the end-to-end process.

We measured the CPU utilization of all the compute nodes in the cluster during the experiment to understand how the system scales in terms of processing data volumes and how the compute tasks are distributed within the cluster. The results of our utilization



■ **Figure 11** Cumulative CPU utilization of compute nodes in the cluster. On the left, a single compute node acts as the ingress point for all of the LiDAR data produced by AGVs. The computer “node1” acts as the shared ingress point. On the right, each AGV connects to a distinct compute node.

measurements are presented in Figure 11. The results show that the total compute load is equivalent for both ingress arrangements, but the compute balance across nodes varies. For the shared ingress node arrangement, the node acting as the gateway is under significantly more load than the rest of the cluster. For 6 AGVs, the shared ingress node is under sufficient load to cause degradation of responsiveness, as is evident in the latency results of Figure 9.

As can be seen from the figures, incoming data streams can be added to the cluster without significantly affecting the latency. The size of the warehouse is realistic, but even with a cluster with modest computing power, we are able to get reasonable latencies (on average 60 ms). It is important to notice that the latencies are about perceiving the overall situation in the warehouse hall. The individual vehicles may need shorter perception latencies for their internal control.

A more powerful cluster is needed for handling denser LiDAR streams and more vehicles, but our solution uses the internal communication links of a cluster to update an occupancy grid. Using, e.g., vehicle-to-vehicle links for the purpose would be inefficient and slow, as would be using distant cloud computing capacity.

6 Discussion

Instead of handling LiDAR data locally in the AGVs, our design is based on sending the LiDAR data to a fog node. Our main motivation is to enable the use of novel computing intensive methods for shared perception. Recently, methods based on machine learning have improved significantly and gained attention. Such methods are based on having the raw data directly available for processing and massive computing capacity for applying the computationally intensive algorithms (see [18] and [13] for related surveys). We see fog computing as a good solution for such needs. On one hand, it enables the use of complex software solutions on computationally powerful hardware. On the other hand, fog computing nodes can be placed close to the AGVs, which makes short latency times possible.

We have chosen a solution based on 5G-TSN systems as they enable real-time operation of the communication network. Other options for organizing the communication exist (see, e.g., [22] for a survey). Such systems are currently under intense research and development work, but not ready for wide scale experimentation. This has motivated us to use simulation as the primary method for our studies. However, modeling and understanding the behavior

of the related complex software is hard. Software layers abstract the details, and there is the risk that simulation models do not capture the complex dependencies hidden by the abstraction layers. Therefore, we have used a hybrid approach, where such software parts of the system are implemented by using real software running on real hardware. Using a generic fog simulator, e.g. [9], would give a different view into AGV systems.

We have not used computational accelerators in our experimentation. Using computational accelerators for handling LiDAR point data is common. It is also typical to use accelerators in machine learning inference systems. Our intention has been to present an overview of a perception system based on a fog node. Our own prior work [8] indicates that the use of typical computational accelerators further complicates the operation of the systems. In the experimentation presented in this paper, we have used small point clouds instead of having computational accelerators, e.g. GPUs, in the system. Similarly, we have omitted the detailed features and analysis of TSN operation as we have concentrated on the system level properties (for detailed features and analysis of TSN operation see, e.g., [12, 16]).

7 Conclusions

In this paper, we presented our hybrid solution for doing research and development work on intelligent AGV traffic systems. Our solution combines a virtual warehouse with a real cluster acting as a fog computing node close to the warehouse. Our experimentation shows that our implementation of the AGV LiDAR sensor data processing on the cluster is feasible for producing a shared view of the observations done from the sensor data streams.

The occupancy information that we compute on a cluster yields a shared real-time view of an observed situation. Our design is based on using hard real-time methods, but in the experimentation we have used simplifications. We see more detailed analysis of the real-time behavior is an important direction for future research.

By sharing the observations and keeping up history, our design supports fault-tolerance and offers information on the motion of the parties in the warehouse. Motion information is typically needed by the traffic control algorithms that coordinate several vehicles. Also considering the fault-tolerance aspects, there is a need for further research.

To get results on traffic coordination, a shared control algorithm could be implemented using the shared LiDAR observation data available in the cluster. Also, AI systems could be added both to the vehicles and to the management system to understand the interplay between the autonomy of vehicles and coordinated decisions by the traffic controller.

References

- 1 Jack E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1):25–30, 1965. doi:10.1147/sj.41.0025.
- 2 Anton Debner, Jaakko Harjuhahto, and Vesa Hirvisalo. A LiDAR dataset from a virtual warehouse. Aalto University, 2020. URL: <https://github.com/Aalto-ESG/fog-iot-2020-data>.
- 3 Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.
- 4 Gregory Dudek and Michael Jenkin. *Computational principles of mobile robotics*. Cambridge University Press, 2010.
- 5 Janos Farkas, Balasz Varga, György Miklos, and Joachim Sachs. 5G-TSN Integration for Industrial Automation. *Ericsson Technology Review*, 07/2019.
- 6 FlatBuffers. The FlatBuffers website, 2020. URL: <https://google.github.io/flatbuffers/>.
- 7 OpenFog Consortium Architecture Working Group. OpenFog reference architecture for fog computing. *OPFRA001*, 20817:162, 2017.

- 8 Jussi Hanhiova, Teemu Kämäräinen, Sipi Sipilä, Matti Siekkinen, Vesa Hirvisalo, and Antti Ylä-Jääski. Latency and throughput characterization of convolutional neural networks for mobile computer vision. In *Proceedings of the 9th ACM Multimedia Systems Conference (MMSys'18)*, 2018. doi:10.1145/3204949.3204975.
- 9 iFogSim. A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments. URL: <https://github.com/Cloudslab/iFogSim>.
- 10 Vasileios Karagiannis. Compute node communication in the fog: Survey and research challenges. In *Proceedings of the Workshop on Fog Computing and the IoT*, pages 36–40, 2019. doi:10.1145/3313150.3313224.
- 11 Alexander Katriniok, Peter Kleibaum, and Martina Joševski. Distributed model predictive control for intersection automation using a parallelized optimization approach. *IFAC-PapersOnLine*, 50(1):5940–5946, 2017. doi:10.1016/j.ifacol.2017.08.1492.
- 12 Dorin Maxim and Ye-Qiong Song. Delay Analysis of AVB traffic in Time-Sensitive Networks (TSN). In *Proceedings Real-Time Networks and Systems (RTNS'17)*, 2017. doi:10.1145/3139258.3139283.
- 13 Ruben Mayer and Hans-Arno Jacobsen. Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques, and Tools. *ACM Computing Surveys*, 53(1), February 2020. doi:10.1145/3363554.
- 14 Mohamed W. Mehrez, Tobias Sprodowski, Karl Worthmann, George K.I. Mann, Raymond G. Gosine, Juliana K. Sagawa, and Jürgen Pannek. Occupancy grid based distributed MPC for mobile robots. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4842–4847, September 2017. doi:10.1109/IROS.2017.8206360.
- 15 Nima Mohajerin and Mohsen Rohani. Multi-step prediction of occupancy grid maps with recurrent neural networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10592–10600, June 2019. doi:10.1109/CVPR.2019.01085.
- 16 Ahmed Nasrallah, Akhilesh S. Thyagaturu, Cuixiang Wang Ziyad Alharbi, Xing Shao, Martin Reisslein, and Hesham Elbakoury. Performance Comparison of IEEE 802.1 TSN Time Aware Shaper (TAS) and Asynchronous Traffic Shaper (ATS). *IEEE Access*, 7, April 2019. doi:10.1109/ACCESS.2019.2908613.
- 17 Arne Neumann, Lukasz Wisniewski, Torsten Musiol, Christian Mannweiler, Borislava Gajic, Rakash SivaSiva Ganesan, and Peter Ros. Abstraction models for 5G mobile networks integration into industrial networks and their evaluation. In *In Kommunikation und Bildverarbeitung in der Automation (Technologien für die intelligente Automation) 12*, 2020. doi:10.1007/978-3-662-59895-5_7.
- 18 Giang Nguyen, Stefan Dlugolinsky, Martin Bobák, Viet Tran, Álvaro López García, Ignacio Heredia, Peter Malík, and Ladislav Hluchý. Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey. *Artificial Intelligence Review*, 52(1):77–124, 2019. doi:10.1007/s10462-018-09679-z.
- 19 Lorenzo Sabattini, Elena Cardarelli, Valerio Digani, Cristian Secchi, Cesare Fantuzzi, and Kay Fuerstenberg. Advanced sensing and control techniques for multi agv systems in shared industrial environments. In *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–7, September 2015. doi:10.1109/ETFA.2015.7301488.
- 20 Shaik Mohammed Salman, Vaclav Struhar, Alessandro V. Papadopoulos, Moris Behnam, and Thomas Nolte. Fogification of industrial robotic systems: Research challenges. In *Proceedings of the Workshop on Fog Computing and the IoT*, IoT-Fog '19, page 41–45, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313150.3313225.
- 21 The HDF Group. Hierarchical data format version 5. URL: <http://www.hdfgroup.org/HDF5>.
- 22 Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330, September 2019. doi:10.1016/j.sysarc.2019.02.009.

Addressing the Node Discovery Problem in Fog Computing

Vasileios Karagiannis 

Distributed Systems Group, TU Wien, Austria
v.karagiannis@dsg.tuwien.ac.at

Nitin Desai 

Mälardalen University, Västerås, Sweden
nitin.desai@mdh.se

Stefan Schulte 

Distributed Systems Group, TU Wien, Austria
s.schulte@dsg.tuwien.ac.at

Sasikumar Punnekkat 

Mälardalen University, Västerås, Sweden
sasikumar.punnekkat@mdh.se

Abstract

In recent years, the Internet of Things (IoT) has gained a lot of attention due to connecting various sensor devices with the cloud, in order to enable smart applications such as: smart traffic management, smart houses, and smart grids, among others. Due to the growing popularity of the IoT, the number of Internet-connected devices has increased significantly. As a result, these devices generate a huge amount of network traffic which may lead to bottlenecks, and eventually increase the communication latency with the cloud. To cope with such issues, a new computing paradigm has emerged, namely: fog computing. Fog computing enables computing that spans from the cloud to the edge of the network in order to distribute the computations of the IoT data, and to reduce the communication latency. However, fog computing is still in its infancy, and there are still related open problems. In this paper, we focus on the node discovery problem, i.e., how to add new compute nodes to a fog computing system. Moreover, we discuss how addressing this problem can have a positive impact on various aspects of fog computing, such as fault tolerance, resource heterogeneity, proximity awareness, and scalability. Finally, based on the experimental results that we produce by simulating various distributed compute nodes, we show how addressing the node discovery problem can improve the fault tolerance of a fog computing system.

2012 ACM Subject Classification Computer systems organization → Cloud computing; Computer systems organization → Fault-tolerant network topologies

Keywords and phrases Fog computing, Edge computing, Internet of Things, Node discovery, Fault tolerance

Digital Object Identifier 10.4230/OASICS.Fog-IoT.2020.5

Funding The research leading to this paper has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785, FORA – Fog Computing for Robotics and Industrial Automation.

1 Introduction

The IoT paradigm envisions a world in which everyday objects (i.e., wearables, dumpsters, phones, etc.) connect to the Internet [14]. Such objects may use this connectivity to exchange, store, and process data in order to sense and to affect the surrounding environment [12]. Since the computational resources of the everyday objects alone may not be sufficient for handling the required computational efforts to achieve this, the IoT devices commonly make use of cloud-based computational resources [24].



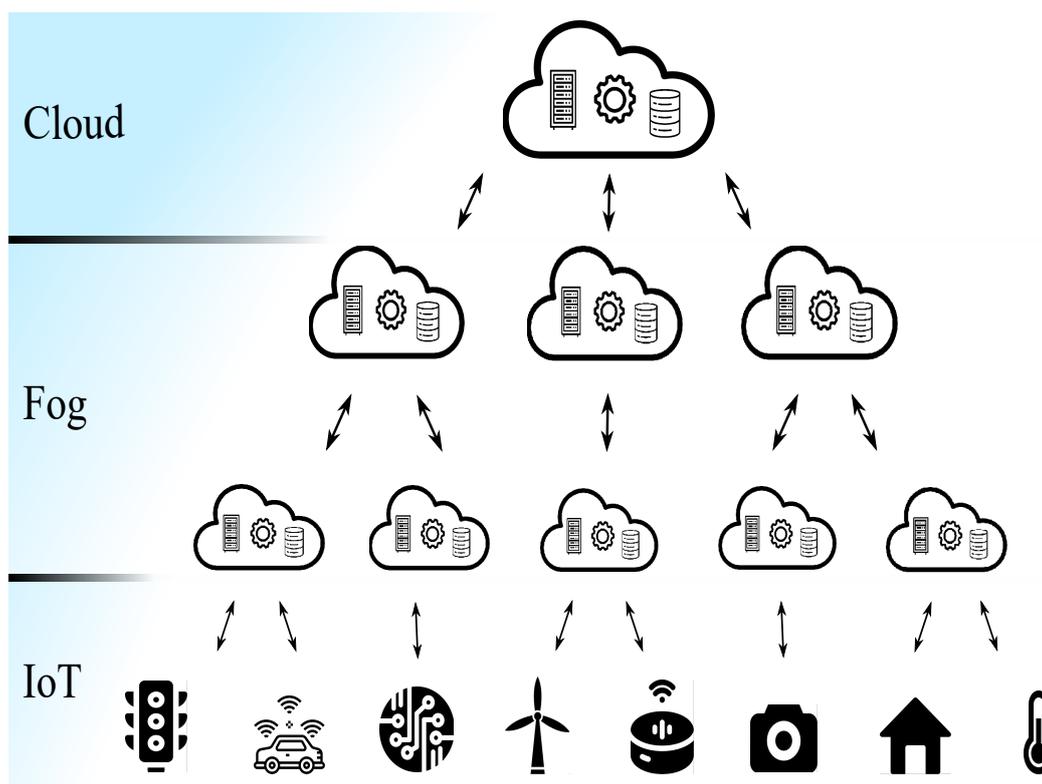
© Vasileios Karagiannis, Nitin Desai, Stefan Schulte, and Sasikumar Punnekkat;
licensed under Creative Commons License CC-BY
2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020).

Editors: Anton Cervin and Yang Yang; Article No. 5; pp. 5:1–5:10

OpenAccess Series in Informatics



OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** A fog computing system consisting of various compute nodes that span from the cloud to the edge of the network.

However, despite the aid of the cloud, the traffic from a large number of Internet-connected devices can still lead to bottlenecks which increase the communication latency, and may even limit the expansion of the IoT [19]. Moreover, there are concerns related to preserving the privacy of the aggregated IoT data, and reducing the communication cost [20]. To cope with such issues, novel computing paradigms have emerged, two of the most popular being fog computing, and edge computing.

One distinguishing characteristic to separate fog computing from edge computing, is that the fog envisions a hierarchy of computational resources which span from the cloud to the edge of the network [3]. For example, Fig. 1 shows a fog computing system that includes various interconnected cloud and fog compute nodes which spread to the network edge where the IoT devices reside. Edge computing on the other hand, aims at pushing the computations towards the edge of the network wherever there are available computational resources (e.g., cloudlets or fog nodes) without explicitly including interactions with the cloud [25].

The research efforts applied in the context of these two paradigms have resulted in architectures, models, and frameworks for performing computations in the proximity of the IoT devices. Due to such efforts, fog computing and edge computing systems have been observed to provide significant benefits for use cases like data stream processing [5], preserving privacy in the IoT [20], performing analytics of IoT data [1], online storage [21], and others [17].

To implement such architectures, compute nodes are provisioned at strategic positions throughout the network in order to distribute the computations, avoid bottlenecks, and reduce the communication latency [13]. A lot of research has been conducted in this context,

resulting in multiple computing systems which aim at leveraging the edge of the network in order to satisfy the application requirements (e.g., regarding latency and bandwidth) and to improve the user experience [15].

Despite the popularity of fog computing and edge computing in the distributed systems research community, computing at the edge of the network is still a relatively recent research topic. For this reason, there are still various important open research problems and challenges, which require further investigation [22]. In this paper, we focus on the node discovery problem [4, 23].

Typically, fog computing and edge computing research assumes that compute nodes are already discovered and integrated in the system [11]. However, this can be a complicated task because the current node discovery approaches usually used in cloud-based systems, are not applicable to fog computing since the problem is very different when dealing with compute nodes at the edge of the network [29]. For instance, fog computing systems are expected to leverage on the proximity of the compute nodes while also considering compute nodes with very diverse resource capacities. Such aspects which have not been considered in the context of cloud computing, make novel node discovery techniques—tailored to fog computing—necessary. For this reason, in this paper we analyze the node discovery problem in fog computing, and we discuss the various related aspects that need to be taken into account. Furthermore, we identify the related research questions which need to be addressed, in order to tackle this problem efficiently.

The rest of this paper is organized as follows: Section 2 discusses related work from the literature. Afterwards, in Section 3, we analyze the node discovery problem in fog computing, and we identify related research questions. Subsequently in Section 4, we present the preliminary evaluation results that we produce based on simulations which show some of the benefits of addressing the proposed problem (regarding fault tolerance). Finally, Section 5 concludes this work, and describes our plans for further research on this topic.

2 Related work

The majority of related work, assumes that the various compute nodes of a fog computing system are already discovered and integrated in the system [11]. Typically, these systems follow a hierarchical architecture whereby the nodes are organized in layers [26]. For instance, Bellavista et al. [2] discuss the execution of services on compute nodes at the edge of the network using a three-layer architecture, and Deng et al. [6] discuss the provisioning of services in distributed edge nodes. However, none of these approaches discuss how the compute nodes are discovered and placed in appropriate positions in the hierarchy.

Kolcun et al. [18] present a distributed platform that allows IoT devices from wireless sensor networks, to send data to cloud and local compute nodes. By shifting the computations from the cloud to the local nodes, this approach reduces the network traffic. Furthermore, the authors propose a node discovery algorithm which aids in finding an appropriate compute node for each IoT device.

Similarly, Tomar and Matam [27] present a framework that allows the data from the IoT devices to be processed in local compute nodes thereby lowering the dependency on the cloud. This framework also includes a node discovery algorithm for finding appropriate compute nodes for the IoT devices.

Finally, Venanzi et al. [31] address the same problem of node discovery for IoT devices although, the focus of this approach is to prolong the lifespan of these devices by considering energy efficiency aspects.

Notably, these approaches focus on the problem of selecting appropriate compute nodes for processing the IoT data. In contrast, the work at hand focuses on the problem of discovering new compute nodes that join a fog computing system. Even though these problems seem similar, they require different solutions. The former problem relies on the wireless communication of the IoT devices to discover potential compute nodes (i.e. the compute nodes that reside within wireless range). In the latter problem, which is the problem we address in our work, the compute nodes that span from the cloud to the edge of the network may not integrate wireless communication. Therefore, the aforementioned solutions that address the node discovery problem in the IoT, do not apply to the node discovery problem in fog computing.

Further related work can be found in approaches that aim at creating fog computing systems for handling applications related to safety. For instance, Dobrin et al. [8] discuss safety-critical applications while focusing on the problem of having unexpected failures, and Desai et al. [7] discuss various safety aspects (with a focus on safety-critical applications) that need to be considered in fog computing systems.

In our work, we also address fault tolerance. However, these works consider fault tolerance as an independent problem which makes it hard to cope with. In our work, we consider fault tolerance at a very early stage, i.e., during the node discovery phase, which increases our options regarding finding appropriate solutions, and based on this, we present promising results.

Therefore, the papers discussed so far either briefly mention the node discovery problem in fog computing, or assume that the compute nodes are already discovered and integrated in the system. Thus, they do not provide an analysis of the problem, or any concrete ways to solve it. On the contrary, in our work we analyze different aspects of this problem, we propose related research questions, and we also present promising results towards addressing the node discovery problem in fog computing efficiently.

3 The Node Discovery Problem

The node discovery problem refers to the way that new compute nodes are detected by the system, as well as the process of integrating these nodes (this is also referred to as the discovery phase). For instance, in Fig. 1 we show a fog computing system consisting of one cloud compute node, and eight fog compute nodes (e.g., cloudlets, base stations, routers, etc.), which are organized in three layers. If a new compute node becomes available, how is this node detected by the system, and with which nodes should the new node communicate? In other words, where should the new node be placed in the hierarchy. There are several options because a new node can be placed in each one of the three layers, and connect to different nodes from the adjacent layers. However, every option has a different impact on the performance of the system. Since fog computing systems are expected to scale massively [9], new compute nodes are likely to join the system frequently. Thus, node discovery is an essential part of fog computing systems.

To address this problem, we analyze the different aspects of a fog computing system that are affected by the manner whereby nodes are discovered and integrated in the system. To this end, the following sections discuss the reason that the node discovery problem affects different aspects of fog computing, and why these aspects are important. Specifically, Section 3.1 discusses fault tolerance, Section 3.2 addresses the potential resource heterogeneity of the nodes, Section 3.3 discusses the importance of proximity awareness, and Section 3.4 addresses scalability. Finally, Section 3.5 presents the research questions that need to be answered in order to address the node discovery problem efficiently.

3.1 Fault Tolerance

In fog computing, some of the participating compute nodes may be unreliable, and might fail unexpectedly at any moment, which can divide a fog computing system into disjoint parts [16], and affect the system's reliability [32]. For this reason, mechanisms for handling node failure become essential. However, this can be especially challenging in fog computing because when a node fails, moving the computations to neighbor nodes or to the cloud, may affect the performance of the system (e.g., might increase the communication latency) [30].

Nevertheless, it is possible to cope with this problem by integrating efficient mechanisms for handling potential future node failures, at the discovery phase, i.e., when a new node joins the system. This can be achieved by having each new node store additional nodes which may not reside in proximity, and are not necessarily used for processing the IoT data, but can be used for maintaining connectivity in case the neighbors fail (cf. Section 4).

3.2 Resource Heterogeneity

Fog computing systems consist of various resource-heterogeneous compute nodes [28]. This means that the participating compute nodes may have very different resource capacities, e.g., regarding CPU and memory, but they may also have different capabilities, e.g., regarding hosted services and applications. This diversity should be taken into account during the discovery phase, because different nodes need to be treated differently. For example, upon discovery, a cloud compute node which is able to provide a huge amount of computational resources should go to the top of the hierarchy. This way, the nodes of lower layers will be able to send the IoT data to that node (for processing) by forwarding the data upwards the hierarchy (cf. Fig 1). On the contrary, a compute node at the edge of the network should be placed close to the IoT devices (cf. Fig 1) in order to leverage on the low communication latency. Therefore, the resource heterogeneity of the compute nodes needs to be considered during the discovery phase in order to ensure the efficient operation of a fog computing system.

3.3 Proximity Awareness

Since processing data in nearby compute nodes improves the communication efficiency [9], fog computing systems leverage on the proximity among the various compute nodes, and the IoT devices, in order to process the IoT data with low communication latency. Most approaches assume that the participating compute node are already discovered and integrated in the system based on proximity (as discussed in Section 1). However, in order to take into account the proximity among the nodes, new nodes need to take proximity measurements (e.g., using round-trip time or hop count), and then connect to the neighbors of the closest proximity.

Taking into account the proximity among the nodes during the discovery phase is a challenging task in fog computing, because proximity measurements may have conflicts with other aspects, e.g., with the resource heterogeneity aspect (cf. Section 3.2). This can happen for instance, upon discovery of a new compute node which integrates a big amount of computational resources, and should be placed in a high layer so that many nodes of lower layers can use these resources. At the same time, this new node may be in the proximity of nodes in lower layers. This means that according to proximity, the new node should be placed in a low layer. Thus, during the discovery phase, there may be conflicts based on the different goals of the discovery problem.

3.4 Scalability

As discussed in Section 1, fog computing systems can include compute nodes that span from the cloud to the edge of the network and thus, they may need to scale to a large degree [9]. This means that during the discovery phase, there can be a huge number of possible positions for a new node. Examining all the possible options means taking proximity measurements for a very large number of potential neighbors. However, this may not be possible since this process generates a considerable amount of network traffic which is part of the overhead of the discovery phase. Furthermore, more messages need to be exchanged in order to discover and store additional nodes for fault tolerance, and in order to examine the resource heterogeneity of the other nodes, as discussed in Sections 3.1 and 3.2. Since generating a significant amount of overhead can compromise the scalability of the system, the overhead of the discovery phase needs to be considered, especially because in fog computing new compute nodes may be discovered at any time [16].

3.5 Research Questions

There are many aspects of fog computing that can be improved by considering the node discovery problem (cf. Sections 3.1 – 3.4). For this reason, and in order to be able to solve this problem efficiently, we identify the following research questions (RQ):

RQ1 To what degree can fog computing systems be fault-tolerant, by storing additional nodes during the discovery phase, which are used in case of node failures?

RQ2 How should the proximity and the resource heterogeneity of the compute nodes, affect the position of a new node that joins a fog computing system?

RQ3 How to make sure that the overhead from new compute nodes joining, does not compromise the scalability of a fog computing system?

When we are able to answer these research questions, then we will be in the position to design efficient discovery mechanisms that aid in improving various aspects of fog computing.

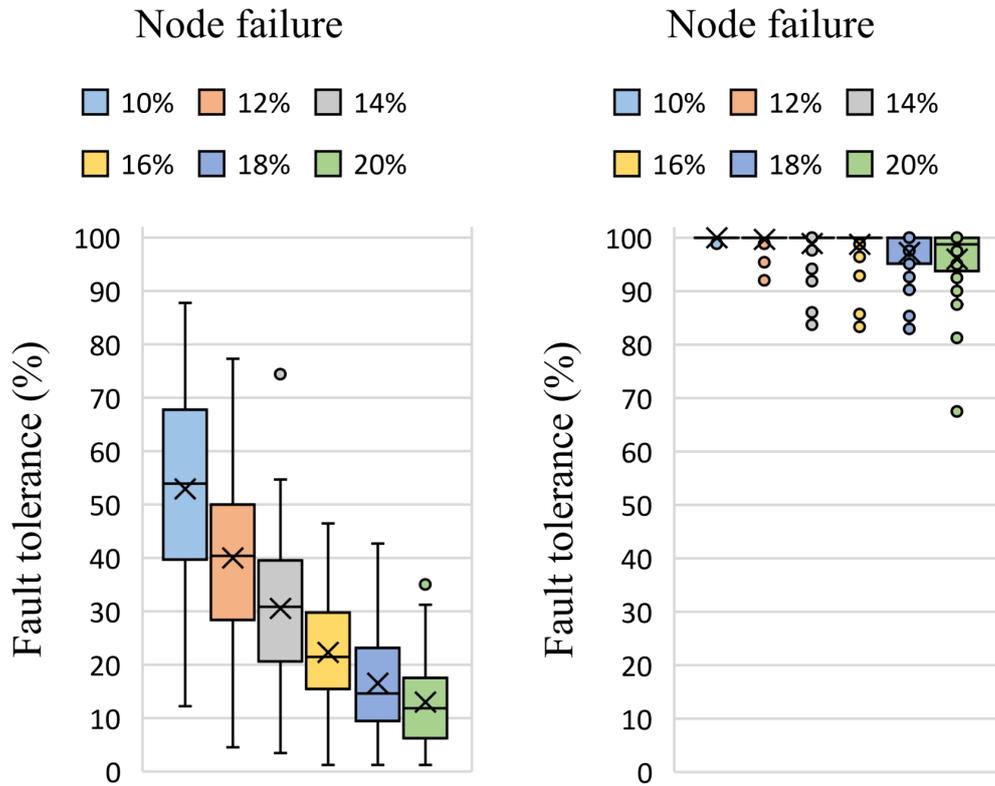
4 Evaluation

In this section, we report the preliminary results of our efforts to tackle the node discovery problem in fog computing. The setup we use in order to produce these results is described in Section 4.1. Afterwards in Section 4.2, we perform various experiments which focus on the fault tolerance aspect of the node discovery problem, and we present our results.

4.1 Evaluation Setup

In order to perform experiments, and examine the fault tolerance of a fog computing system, we have built a simulator using Java. The reason we do not use a simulator developed in the scope of related work from the literature (e.g., iFogSim [10]), is that alternative simulators lack the necessary functionality to address the proposed problem (e.g., compute nodes that fail or become unavailable temporarily).

By using our simulator, we are able to simulate hierarchical fog computing systems consisting of compute nodes that span from the cloud to the edge of the network. The number of the participating compute nodes in these systems is configurable, but the layout is always hierarchical. In the hierarchy, every parent node selects as neighbors up to three children nodes, as shown in Fig. 1. For this evaluation, we perform 50 experiments with



(a) When each compute node stores only nearby neighbors. (b) When each compute node stores neighbors and additional nodes to be used in case of failures.

■ **Figure 2** Fault tolerance of a fog computing system.

100 nodes. The reason we have selected these specific numbers, is that after experimenting extensively with this simulator, we found these numbers to produce results which can be considered representative of the general case.

In each one of the 50 experiments, we select various percentages of the participating compute nodes to become unresponsive, and then we examine the percentage of the responsive nodes that remain connected. Since node failure can divide a fog computing system into disjoint parts (as discussed in Section 3.1), with this experiment we aim at measuring the fault tolerance of the system. The specific nodes that fail are chosen randomly using the uniform distribution. Using this evaluation setup, we examine two node discovery mechanisms.

In the first mechanism, each new node requests to join from a preexisting node of the system (i.e., a contact node), and stores only nearby neighbors which are found through the contact node. In the second, the new node requests to join through the contact node again, but apart from storing the nearby neighbors, it also stores the neighbors of the contact node. The neighbors of the contact node may not reside nearby so they might not be suitable for processing data with low communication latency. However, these nodes are used in case the other neighbors fail.

4.2 Evaluation Results

In Fig. 2, we show the results of our experiments. For Fig. 2a, the nodes store only neighbors, i.e., using the first node discovery mechanism (cf. Section 4.1). In this experiment, we induce node failure of 10%, 12%, 14%, 16%, 18%, and 20% of the nodes, and we measure the corresponding percentages of the responsive nodes that remain connected. Each box plot includes 50 values from the 50 experiments we have conducted. Notably, the average percentage of responsive compute nodes that remain connected is approximately 53% with 10% node failure, and the fault tolerance of the system decreases, while the percentage of node failures increases.

For Fig. 2b, we repeat the same experiment, but we change the node discovery mechanism. Instead of storing only neighbors (as done for Fig. 2a), in this experiment every node stores additional nodes to be used in case of failures, i.e., the second node discovery mechanism (cf. Section 4.1). Thus, when a responsive node detects (e.g., using heartbeat messages) that the neighbors have failed, this node tries to connect to the system using the additional nodes. Notably, the average percentage of responsive compute nodes that remain connected in this experiment, is approximately 99% with 10% of node failure. Again, the fault tolerance of the system decreases, while the node failures increase although, until the node failures reach 20%, the average fault tolerance remains always above 90%.

Based on Fig. 2a, we note that creating a fog computing system whereby each node stores only its neighbors, is not an efficient approach with regard to fault tolerance. This is claimed because, when various nodes fail, the percentage of remaining responsive nodes which remain connected decreases radically.

However, according to Fig. 2b, we note that the fault tolerance of a fog computing system can be increased significantly, by storing additional nodes during the node discovery phase. Similarly, we believe that addressing the node discovery problem can aid in improving various aspects of fog computing systems, as discussed in Section 3.

5 Conclusion

In this paper, we present the node discovery problem in fog computing systems. To this end, we analyze various aspects of fog computing that can be affected from the way new nodes are discovered and integrated in the system, such as: fault tolerance, resource heterogeneity, proximity awareness, and scalability. Furthermore, we identify related research questions which need to be addressed in order to tackle the proposed problem efficiently. Finally, we simulate fog computing systems, and we perform experiments with various compute nodes which integrate a node discovery mechanism that focuses on improving the fault tolerance of the system. By analyzing the results, we show that when each new node that joins, stores additional nodes during the discovery phase, the fault tolerance of a fog computing system improves significantly.

Due to the promising results, in the future we plan to focus on node discovery mechanisms that improve fog computing systems. Specifically, we plan to design node discovery mechanisms tailored to fog computing systems by considering not only the fault tolerance of the system, but also aspects related to proximity awareness, resource heterogeneity, scalability, and others.

References

- 1 Hamid Reza Arkian, Abolfazl Diyanat, and Atefe Pourkhalili. Mist: Fog-based data analytics scheme with cost-efficient resource provisioning for iot crowdsensing applications. *Journal of Network and Computer Applications*, 82:152–165, 2017.

- 2 Paolo Bellavista, Alessandro Zanni, and Michele Solimando. A migration-enhanced edge computing support for mobile devices in hostile environments. In *International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 957–962. IEEE, 2017.
- 3 Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Workshop on Mobile Cloud Computing (MCC)*, pages 13–16. ACM, 2012.
- 4 Rustem Dautov, Salvatore Distefano, Dario Bruneo, Francesco Longo, Giovanni Merlino, Antonio Puliafito, and Rajkumar Buyya. Metropolitan intelligent surveillance systems for urban areas by harnessing iot and edge computing paradigms. *Software: Practice and Experience*, 48(8):1475–1492, 2018.
- 5 Marcos Dias de Assuncao, Alexandre da Silva Veith, and Rajkumar Buyya. Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. *Journal of Network and Computer Applications*, 103:1–17, 2018.
- 6 Shuiguang Deng, Zhengzhe Xiang, Jianwei Yin, Javid Taheri, and Albert Y Zomaya. Composition-driven iot service provisioning in distributed edges. *IEEE Access*, 6:54258–54269, 2018.
- 7 Nitin Desai and Sasikumar Punnekkat. Safety of fog-based industrial automation systems. In *Workshop on Fog Computing and the IoT (IoT-Fog)*, pages 6–10. ACM, 2019.
- 8 Radu Dobrin, Nitin Desai, and Sasikumar Punnekkat. On fault-tolerant scheduling of time sensitive networks. In *Workshop on Security and Dependability of Critical Embedded Real-Time Systems (CERTS)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
- 9 Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. Edge-centric computing: Vision and challenges. *ACM SIGCOMM Computer Communication Review*, 45(5):37–42, 2015.
- 10 Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296, 2017.
- 11 Cheol-Ho Hong and Blesson Varghese. Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms. *ACM Computing Surveys (CSUR)*, 52(5):1–37, 2019.
- 12 Vasileios Karagiannis. Building a Testbed for the Internet of Things. *Alexander Technological Educational Institute of Thessaloniki*, pages 1–92, 2014.
- 13 Vasileios Karagiannis. Compute node communication in the fog: Survey and research challenges. In *Workshop on Fog Computing and the IoT (IoT-Fog)*, pages 1–5. ACM, 2019.
- 14 Vasileios Karagiannis, Periklis Chatzimisios, Francisco Vazquez-Gallego, and Jesus Alonso-Zarate. A survey on application layer protocols for the internet of things. *ICAS Transaction on IoT and Cloud Computing*, 3(1):11–17, 2015.
- 15 Vasileios Karagiannis and Apostolos Papageorgiou. Network-integrated edge computing orchestrator for application placement. In *International Conference on Network and Service Management (CNSM)*, pages 1–5. IEEE, 2017.
- 16 Vasileios Karagiannis, Stefan Schulte, Joao Leitao, et al. Enabling fog computing using self-organizing compute nodes. In *International Conference on Fog and Edge Computing (ICFEC)*, pages 1–10. IEEE, 2019.
- 17 Vasileios Karagiannis, Alexandre Venito, Rodrigo Coelho, Michael Borkowski, and Gerhard Fohler. Edge computing with peer to peer interactions: Use cases and impact. In *Workshop on Fog Computing and the IoT (IoT-Fog)*, pages 1–5. ACM, 2019.
- 18 Roman Kolcun, David Boyle, and Julie A McCann. Optimal processing node discovery algorithm for distributed computing in iot. In *2015 5th International Conference on the Internet of Things (IOT)*, pages 72–79. IEEE, 2015.

- 19 Yang Liu, Jonathan E Fieldsend, and Geyong Min. A framework of fog computing: Architecture, challenges, and optimization. *IEEE Access*, 5:25445–25454, 2017.
- 20 Rongxing Lu, Kevin Heung, Arash Habibi Lashkari, and Ali Akbar Ghorbani. A lightweight privacy-preserving data aggregation scheme for fog computing-enhanced iot. *IEEE Access*, 5:3302–3312, 2017.
- 21 Ivan Lujic, Vincenzo De Maio, and Ivona Brandic. Efficient edge storage management based on near real-time forecasts. In *International Conference on Fog and Edge Computing (ICFEC)*, pages 21–30. IEEE, 2017.
- 22 Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H. Glitho, Monique J. Morrow, and Paul A. Polakos. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 20(1):416–464, 2017.
- 23 Ilir Murturi, Cosmin Avasalcari, Christos Tsigkanos, and Schahram Dustdar. Edge-to-edge resource discovery using metadata replication. In *International Conference on Fog and Edge Computing (ICFEC)*, pages 1–6. IEEE, 2019.
- 24 Carlo Puliafito, Enzo Mingozzi, Francesco Longo, Antonio Puliafito, and Omer Rana. Fog computing for the internet of things: A survey. *ACM Transactions on Internet Technology*, 19(2):18, 2019.
- 25 Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.
- 26 Vitor Barbosa Souza, Xavi Masip-Bruin, Eva Marín-Tordera, Sergi Sánchez-López, Jordi Garcia, Guang-Jie Ren, Admela Jukan, and Ana Juan Ferrer. Towards a proper service placement in combined fog-to-cloud (F2C) architectures. *Future Generation Computer Systems*, 87:1–15, 2018.
- 27 Nitendra Tomar and Rakesh Matam. Optimal query-processing-node discovery in iot-fog computing environment. In *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 237–241. IEEE, 2018.
- 28 Luis M Vaquero and Luis Roderó-Merino. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, 2014.
- 29 Blesson Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick, and Dimitrios S Nikolopoulos. Challenges and opportunities in edge computing. In *International Conference on Smart Cloud (SmartCloud)*, pages 20–26. IEEE, 2016.
- 30 Prateeksha Varshney and Yogesh Simmhan. Demystifying fog computing: Characterizing architectures, applications and abstractions. In *International Conference on Fog and Edge Computing (ICFEC)*, pages 115–124. IEEE, 2017.
- 31 Riccardo Venanzi, Burak Kantarci, Luca Foschini, and Paolo Bellavista. MQTT-driven node discovery for integrated IoT-fog settings revisited: The impact of advertiser dynamicity. In *Symposium on Service-Oriented System Engineering (SOSE)*, pages 31–39. IEEE, 2018.
- 32 Zhenyu Wen, Renyu Yang, Peter Garraghan, Tao Lin, Jie Xu, and Michael Rovatsos. Fog orchestration for internet of things services. *IEEE Internet Computing*, 21(2):16–24, 2017.

Routing Using Safe Reinforcement Learning

Gautham Nayak Seetanadi 

Department of Automatic Control, Lund University, Sweden
gautham@control.lth.se

Karl-Erik Årzén 

Department of Automatic Control, Lund University, Sweden
karlerik@control.lth.se

Abstract

The ever increasing number of connected devices has led to a meteoric rise in the amount of data to be processed. This has caused computation to be moved to the edge of the cloud increasing the importance of efficiency in the whole of cloud. The use of this fog computing for time-critical control applications is on the rise and requires robust guarantees on transmission times of the packets in the network while reducing total transmission times of the various packets.

We consider networks in which the transmission times that may vary due to mobility of devices, congestion and similar artifacts. We assume knowledge of the worst case transmission times over each link and evaluate the typical transmission times through exploration. We present the use of reinforcement learning to find optimal paths through the network while never violating preset deadlines. We show that with appropriate domain knowledge, using popular reinforcement learning techniques is a promising prospect even in time-critical applications.

2012 ACM Subject Classification Computing methodologies → Reinforcement learning; Networks → Packet scheduling

Keywords and phrases Real time routing, safe exploration, safe reinforcement learning, time-critical systems, dynamic routing

Digital Object Identifier 10.4230/OASICS.Fog-IoT.2020.6

Funding The authors are members of the LCCC Linnaeus Center and the ELLIIT Strategic Research Area at Lund University. This work was supported by the Swedish Research Council through the project “Feedback Computing”, VR 621-2014-6256.

1 Introduction

Consider a network of devices in a smart factory. Many of the devices are mobile and communicate with each other on a regular basis. As their proximity to the other devices change, the communication delays experienced by the device also change. Using static routing for such time-critical communications leads to pessimistic delay bounds and underutilization of network infrastructure.

Recent work [2] proposes an alternate model for representing delays in such time-critical networks. Each link in a network has delays that can be characterised by a conservative upper bound on the delay and the typical delay on the link. This dual representation of delay allows for capturing the communication behavior of different types of devices.

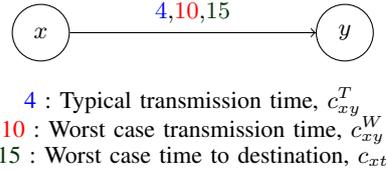
For example, a communication link between two stationary devices can be said to have equal typical and worst case delays. A device moving on a constant path near another stationary device can be represented using a truncated normal distribution. Adaptive routing techniques are capable of achieving smaller typical delays in such scenarios compared to static routing.



© Gautham Nayak Seetanadi and Karl-Erik Årzén;
licensed under Creative Commons License CC-BY
2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020).
Editors: Anton Cervin and Yang Yang; Article No. 6; pp. 6:1–6:8
OpenAccess Series in Informatics



OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Each link with attributes.

The adaptive routing technique described from [2] uses both delay information (typical and worst case) to construct routing tables. Routing is then accomplished using the tables which consider typical delays to be deterministic. This is however not the case as described above.

We propose using Reinforcement Learning (RL) [8, 12] for routing packets. RL is a model-free machine learning algorithm that has found prominence in the field of AI given its light computation and promising results [5, 9]. RL agents learn by exploring the environment around them and then obtaining a reward at the end of one iteration denoted one episode.

RL has been proven to be very powerful but it has some inherent drawbacks when considering its application to time-critical control applications. RL requires running a large number of episodes for an agent to learn. This leads to the possibility of deadline violations during exploration. Another drawback is the large state-space used for learning in classical RL methods that leads to complications in storage and search.

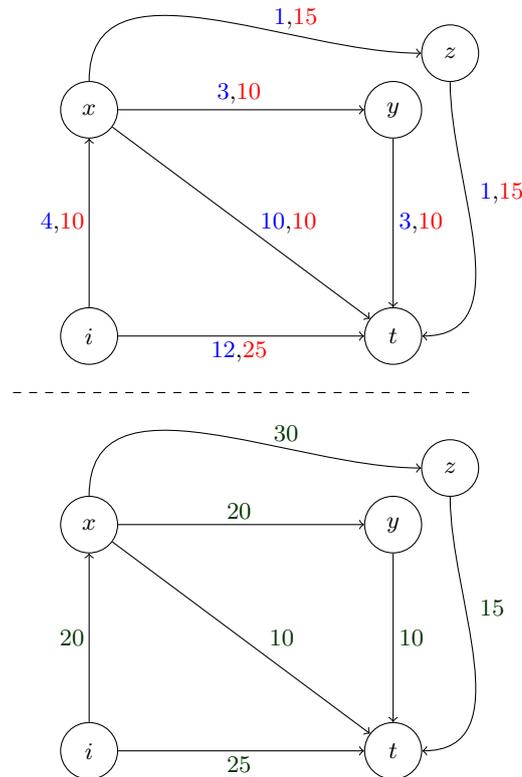
In this paper, we augment classical reinforcement learning with safe exploration to perform *safe* reinforcement learning. We use a simple (Dijkstras[4]) algorithm to perform safe exploration and then use the obtained information for safe learning. Using methodology described in Section 4 we show that safety can be guaranteed during the exploration phase. Using safe RL also restricts the state-space reducing its size. Our decentralised algorithm allows each agent/node in the network to make independent decisions further reducing the state space. *Safe* reinforcement learning explores the environment to dynamically sample typical transmission times and then reduce delays for future packet transmissions. Our Decentralised approach allows each node to make independent and safe routing decisions irrespective of future delays that might be experienced by the packet.

2 System Architecture

Consider a network of nodes, where each link $e : (x \rightarrow y)$ between node x and y is described by delays as shown in Figure 1.

- **Worst case delay (c_{xy}^W):** The delay that can be guaranteed by the network over each link. This is never violated even under maximum load.
- **Typical delay (c_{xy}^T):** The delay that is encountered when transmitting over the link and varies for each packet. We assume this information to be hidden from the algorithm and evaluated by sampling the environment.
- **Worst case delay to destination (c_{xt}):** The delay that can be guaranteed from node x to destination t . Obtained after the pre-processing described in Section 4.1.

A network of devices and communication links can be simplified as a Directed Acyclic Graph as shown in Figure 2. The nodes denote the different devices in the network and the links denote the connections between the different devices. For simplicity we only assume one-way communication and consider a scenario of transmitting a packet from an edge device i to a server, t at a location far away from it.



■ **Figure 2** Example of graph and corresponding state space for the reinforcement learning problem formulation.

As seen from the graph, many paths exist from the source i to t destination that can be utilised depending upon the deadline D_F of the packet.

The values of c_{xy}^T and c_{xy}^W are shown in blue and red respectively for each link $e(x \rightarrow y)$. We also show the value of c_{xt} in green obtained after the pre-processing stage described in the following section.

3 Reinforcement Learning

Reinforcement Learning is the area of machine learning dealing with teaching agents to learn by performing actions to maximise a reward obtained [8] RL generally learns the environment by performing actions (safe actions) and evaluating the reward obtained at the end of the episode. We use Temporal-Difference (TD) methods for estimating state values and discover optimal paths for packet transmission.

We model our problem of transmitting packets from source i to destination t as a Markov Decision Process (MDP) as is the standard in RL. An MDP is a 4-tuple $(\mathcal{S}, \mathcal{A}, P, R)$, where \mathcal{S} is a set of finite states, \mathcal{A} is a set of actions, $P : (s, a, s') \rightarrow \{p \in \mathbb{R} \mid 0 \leq p \leq 1\}$ is a function that encodes the probability of transitioning from state s to state s' as a result of an action a , and $R : (s, a, s') \rightarrow \mathbb{N}$ is a function that encodes the reward received when the choice of action a determines a transition from state s to state s' . We use actions to encode the selection of an outgoing edge from a vertex.

3.1 TD Learning

TD learning [8] is a popular reinforcement learning algorithm that gained popularity due to its expert level in playing backgammon [9]. This model-free learning uses both state s and action a information to perform actions from the state given by the Q -value, $Q(s, a)$. TD learning is only a method to evaluate the value of being in the particular state. It is generally coupled with an exploration policy to form the strategy for an agent. We use a special TD learning called one step TD learning that allows for decentralised learning and allows for each node to make independent routing decisions. The value update policy is given by

$$Q(s, a) = Q(s, a) + \alpha \cdot (\mathcal{R} + \max(\gamma Q(s', a')) - Q(s, a)) \quad (1)$$

3.2 Exploration Policy

ϵ -greedy exploration algorithm ensures that the optimal edge is chosen for most of the packet transmissions while at the same time other edges are explored in search of a path with higher reward. The chosen action $a \in A$ is either one that has the max value V or is a random action that explores the state space. The policy explores the state space with a probability ϵ and the most optimal action is taken with the probability $(1 - \epsilon)$. Generally the value of ϵ is small such that the algorithm exploits the obtained knowledge for most of the packet transmissions. To ensure that the deadline D_F is never violated, we modify the exploration phase to ensure safety and perform *safe* reinforcement learning.

4 Algorithm

We split our algorithm into two distinct phases. A pre-processing phase that gives us the initial safe bounds required to perform safe exploration. A run-time phase then routes packets through the network.

At each node, the algorithm explores feasible paths. During the initial transmissions the typical transmission times are evaluated after packet transmission. During the following transmissions, the path with the least delay is chosen more frequently while also exploring new feasible paths for lower delays. All transmissions using our algorithm are guaranteed to never violate any deadlines as we use safe exploration.

4.1 Pre-processing Phase

The pre-processing phase determines the safe bound for the worst case delay to destination t from every edge $e : (x \rightarrow y)$ in the network. The algorithm used by our algorithm is very similar to the one in [2]. This is crucial to ensure that there are no deadline violations during exploration in the run-time phase and is necessary irrespective of the run-time algorithm used. Dijkstra's shortest path algorithm [7, 4] is used to obtain these values as shown in Algorithm 1.

4.2 Run-time Phase

The run-time algorithm is run at each node on the arrival of a packet. It determines $e : (x \rightarrow y)$ the edge on which the packet is transmitted from the node x to node y . Then the node y executes the run-time algorithm till the packet reaches the destination.

Algorithm 1 Pre-Processing.

```

1: for each node  $u$  do
2:   for each edge  $(u \rightarrow v)$  do
3:     // Delay bounds as described in Section 4.1
4:      $c_{uv} = c_{uv}^W + \min(c_{vt})$ 
5:     // Initialise the Q values to 0
6:      $Q(u, v) = 0$ 

```

Algorithm 2 Node Logic (u).

```

1: for Every packet do
2:   if  $u = \text{source node } i$  then
3:      $D_u = D_F$  // Initialise the deadline
4:      $\delta_{it} = 0$  // Initialise total delay for packet = 0
5:   for each edge  $(u \rightarrow v)$  do
6:     if  $c_{uv} > D_u$  then // Edge is infeasible
7:        $P(u|v) = 0$ 
8:     else if  $Q(u, v) = \max(Q(u, a \in A))$  then
9:        $P(u|v) = (1 - \epsilon)$ 
10:    else
11:       $P(u|v) = \epsilon / (\text{size}(\mathcal{F}) - 1)$ 
12:    Choose edge  $(u \rightarrow v)$  with  $P$ 
13:    Observe  $\delta_{uv}$ 
14:     $\delta_{it} += \delta_{uv}$ 
15:     $D_v = D_u - \delta_{uv}$ 
16:     $R = \text{Environment Reward Function}(v, \delta_{it})$ 
17:     $Q(u, v) = \text{Value iteration from Equation (1)}$ 
18:    if  $v = t$  then
19:      DONE

```

The edge chosen can be one of two actions:

- **Exploitation action:** An action that chooses the path with the least transmission time out of all known feasible paths. If no information is known on all the edges, then an edge is chosen at random.
- **Exploration action:** An action where a sub-optimal node is chosen to transmit the packet. This action uses the knowledge about c_{xy} obtained during the pre-processing phase to ensure that the exploration is safe. This action ensures that the algorithm is dynamic by ensuring that if there exists a path with lower transmission delay, it will be explored and chosen more during future transmissions. Exploration also optimises for a previously congested edge that could be decongested at a later time.

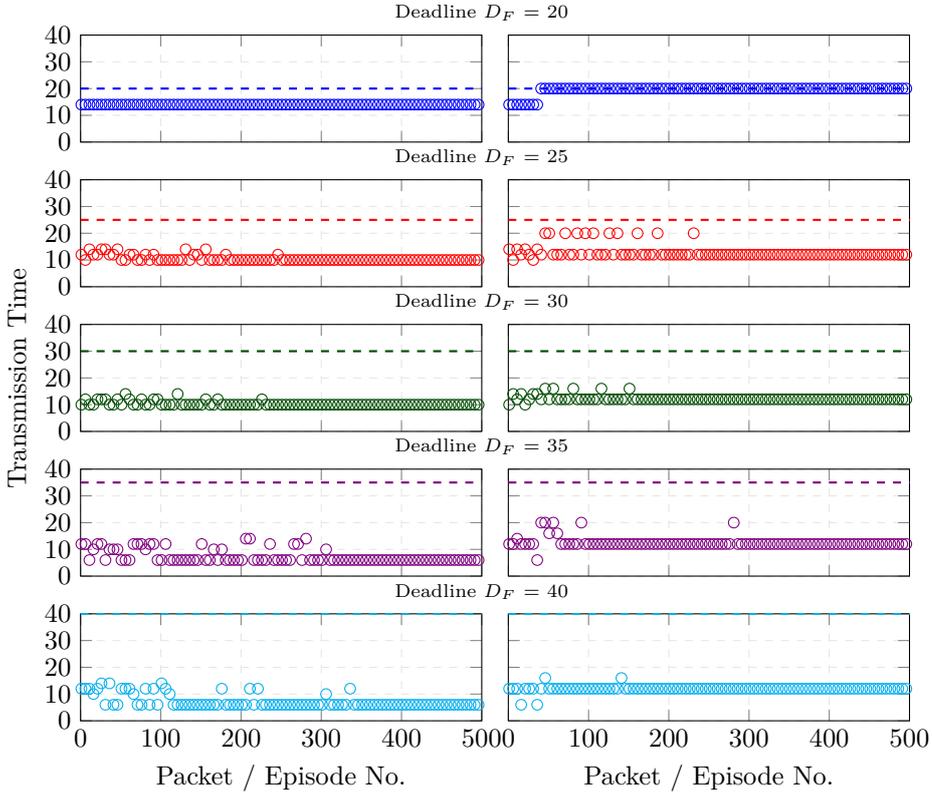
Algorithm 2 shows the pseudo code for the run-time phase. The computation is computationally light and can be run on mobile IoT devices.

4.3 Environment Reward

The reward \mathcal{R} is awarded as shown in Algorithm 3. After each traversal of the edge, the actual time taken δ is recorded and added to the total time traversed for the packet, $\delta_{it} += \delta$. The reward is then awarded at the end of each episode and it is equal to the amount of time saved for the packet, $R = D_F - \delta_{it}$.

■ **Algorithm 3** Environment Reward Function(v, δ_{it}).

-
- 1: Assigns the reward at the end of transmission
 - 2: **if** $v = t$ **then**
 - 3: $R = D_F - \delta_{it}$
 - 4: **else**
 - 5: $R = 0$
-



■ **Figure 3** Smoothed Total Delay for Experiment with (a) Constant delays and (b) Congestion at packet 40.

■ **5 Evaluation**

In this section, we will evaluate the performance of our algorithm. We apply it to the network shown in Figure 2. The network is built using Python and the NetworkX package [6] package. The package allows us to build Directed Acyclic Graphs (DAGs) with custom delays. Each link $e : (x \rightarrow y)$ in the network has the constant worst case link delay c_{xy}^W visible to the algorithm but the value of c_{xy}^T although present is not visible to our algorithm. The pre-processing algorithm and calculates the value of c_{xt} . This is done only once initially and then Algorithms 2 and 3 are run for every packet that is transmitted and records the actual transmission time δ_{it} .

Figure 3 shows the total transmission times when the actual transmission times δ and typical transmission times c^T are equal. We route 500 packets through the network for deadline $D_F \in (20, 25, 30, 35, 40)$. For $D_F = 20$, the only safe path is $(i \rightarrow x \rightarrow t)$ and so has a constant δ_{it} for all packets. For the remaining deadlines, the transmission times vary as

new paths are taken during exploration. The deadlines are never violated for any packets irrespective of the deadline. Table 1 shows the optimal paths and the average transmission times compared to the algorithm from [2].

Figure 3 shows the capability of our algorithm in adapting to congestions in the network. Congestion is added on the link ($i \rightarrow x$) after the transmission of 40 packets. The transmission time over the edge increases from 4 to 10 time units and is kept congested for the rest of the packet transmissions. The algorithm adapts to the congestion by exploring other paths that might now have lower total transmission times δ_{it} . In all cases other than $D_F = 20$, the algorithm converges to the path ($i \rightarrow t$) with $\delta_{it} = 12$. When $D_F = 20$, ($i \rightarrow x \rightarrow t$) is the only feasible path.

6 Practical Considerations

In this section we will discuss some of the practical aspects when implementing the algorithms described in Section 4.

6.1 Computational Overhead

The computational complexity of running our algorithm mainly arises in the pre-processing stage. This complexity is dependent on the number of nodes in the network. Dijkstras algorithm has been widely studied and have efficient implementations that reduce computation. The pre-processing has to be run only once for all networks given that there are no structural changes.

6.2 Multiple sources

The presence of multiple sources and thus multiple packets on the same link can be seen as an increase in the typical delays on the link. This holds true given that the worst case delay c_{xy}^W over each link is properly determined and guaranteed.

6.3 Network changes

- **Node Addition:** During the addition of a new node the pre-processing stage has to be run in a constrained space. The propagation of new information to the preceding nodes is only necessary if it affects the value of c_{xt} over the affected links. The size of the network affected has to be investigated further.
- **Node Deletion:** In the event of node deletion during the presence of a packet at the deleted node, the packet is lost and leads to deadline violation. However no further packages will be transmitted over the link as the reward \mathcal{R} is 0. Similar to the case of node addition, the pre-processing algorithm requires further investigations.

■ **Table 1** Optimal Path for Different Deadlines.

D_F	Optimal Path	Delays [2]	Average Delays (1000 episodes)
15	Infeasible	–	–
20	{i,x,t}	14	14
25	{i,x,y,t}	10	10.24
30	{i,x,y,t}	10	10.22
35	{i,x,z,t}	6	6.64
40	{i,x,z,t}	6	6.55

7 Conclusion and Future Work

In this paper we use *safe* reinforcement learning to routing networks with variable transmission times. A once used pre-processing algorithm is used to determine safe bounds. Then a safe reinforcement learning algorithm uses this domain knowledge to route packets in minimal time with deadline guarantees. We have considered only two scenarios in this paper but we believe that the algorithm will be able to adapt with highly variable transmission times and network failures. The use of low complexity RL algorithm makes it suitable for use on small, mobile platforms.

Although we show stochastic convergence in our results with no deadline violations, our current work lacks **formal guarantees**. Recent work has been published trying to address analytical safety guarantees of safe reinforcement learning algorithms [10, 11]. In [10], the authors perform safe Bayesian optimization with assumptions on Lipschitz continuity of function. While [10] estimates the safety of only one function, our algorithm is dependent on the continuity of multiple functions and requires more investigation.

The network implementation and evaluation using NetworkX in this paper have shown that using *safe* RL is a promising technique. An extension of this work would be implementation on a network emulator. Using network emulators (for example CORE [1], Mininet [3]) would allow us to evaluate the performance of our algorithm on a full internet protocol stack. Using an emulator allows for implementation of multiple flows between multiple sources and destinations.

References

- 1 Jeff Ahrenholz. Comparison of core network emulation platforms. In *2010-Milcom 2010 Military Communications Conference*, pages 166–171. IEEE, 2010.
- 2 Sanjoy Baruah. Rapid routing with guaranteed delay bounds. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 13–22, December 2018.
- 3 Rogério Leão Santos De Oliveira, Christiane Marie Schweitzer, Ailton Akira Shinoda, and Ligia Rodrigues Prete. Using mininet for emulation and prototyping software-defined networks. In *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pages 1–6. Ieee, 2014.
- 4 E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959. doi:10.1007/BF01386390.
- 5 Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sébastien Racanière, Théophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, et al. An investigation of model-free planning. *arXiv preprint*, 2019. arXiv:1901.03559.
- 6 Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11–15, Pasadena, CA USA, 2008.
- 7 Kurt Mehlhorn and Peter Sanders. *Algorithms and Data Structures: The Basic Toolbox*. Springer Publishing Company, Incorporated, 1 edition, 2008.
- 8 Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An Introduction*. Adaptive computation and machine learning. MIT Press, 2018.
- 9 Gerald Tesaro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, March 1995. doi:10.1145/203330.203343.
- 10 Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. Safe exploration for interactive machine learning. In *Proc. Neural Information Processing Systems (NeurIPS)*, December 2019.
- 11 Kim P Wabersich and Melanie N Zeilinger. Safe exploration of nonlinear dynamical systems: A predictive safety filter for reinforcement learning. *arXiv preprint*, 2018. arXiv:1812.05506.
- 12 Marco Wiering and Martijn Van Otterlo. Reinforcement learning. *Adaptation, learning, and optimization*, 12:3, 2012.

Real-Time Containers: A Survey

Václav Struhár 

Mälardalen University, Västerås, Sweden
vaclav.struhar@mdh.se

Moris Behnam

Mälardalen University, Västerås, Sweden
moris.behnam@mdh.se

Mohammad Ashjaei 

Mälardalen University, Västerås, Sweden
mohammad.ashjaei@mdh.se

Alessandro V. Papadopoulos 

Mälardalen University, Västerås, Sweden
alessandro.papadopoulos@mdh.se

Abstract

Container-based virtualization has gained a significant importance in a deployment of software applications in cloud-based environments. The technology fully relies on operating system features and does not require a virtualization layer (hypervisor) that introduces a performance degradation. Container-based virtualization allows to co-locate multiple isolated containers on a single computation node as well as to decompose an application into multiple containers distributed among several hosts (e.g., in fog computing layer). Such a technology seems very promising in other domains as well, e.g., in industrial automation, automotive, and aviation industry where mixed criticality containerized applications from various vendors can be co-located on shared resources.

However, such industrial domains often require real-time behavior (i.e., a capability to meet predefined deadlines). These capabilities are not fully supported by the container-based virtualization yet. In this work, we provide a systematic literature survey study that summarizes the effort of the research community on bringing real-time properties in container-based virtualization. We categorize existing work into main research areas and identify possible immature points of the technology.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Software and its engineering → Virtual machines

Keywords and phrases Real-Time, Containers, Docker, LXC, PREEMPT_RT, Xenomai, RTAI

Digital Object Identifier 10.4230/OASICS.Fog-IoT.2020.7

Funding The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785, FORA – Fog Computing for Robotics and Industrial Automation.

1 Introduction

Fog Computing as well as cloud computing relies extensively on resource virtualization. In this area, the container-based virtualization is gaining its importance as a lightweight alternative of hypervisor-based virtualization. The container technology allows to execute applications and their software dependencies in a virtual environment independently on the software ecosystem of their hosts. A host can accommodate multiple containers at a time, providing means for container isolation and resource control for the containers. Container-based virtualization (sometimes referred as an OS level virtualization) does not require a hypervisor and therefore it provides near-native performance [13, 25], rapid deployment times and a low



© Václav Struhár, Moris Behnam, Mohammad Ashjaei, and Alessandro V. Papadopoulos;
licensed under Creative Commons License CC-BY

2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020).

Editors: Anton Cervin and Yang Yang; Article No. 7; pp. 7:1–7:9

OpenAccess Series in Informatics



OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

overhead while still retaining a certain level of resource isolation and resource control. The containers are a de-facto standard for development of large scale web applications adopted by a number of companies [7].

The benefits of the container-based virtualization are aligned with the strive of the companies in other areas such as in industrial and robot control, automotive and aviation. In these industrial domains, there are strong requirements to (i) consolidate computational resources (Electronic Control Units, physical controllers) and (ii) provide a flexible environment for running (real-time) applications. Additionally, container-based virtualization can enable interruption-free hardware and software maintenance, dynamic system redundancy change and system redundancy healing [16]. However, in such fields stringent real-time requirements are often needed. This means that the applications inside of a container should meet predefined deadlines independently on other co-located containers.

In this survey, we summarize the research carried out in the area of real-time containers since the introduction of containers in Linux (i.e. 2008 [1]).

The main contributions of this paper include:

- Systematic literature survey of the real-time containers.
- Overview of the approaches and technology enabling real-time behavior of containers.
- Identification of pitfalls, challenges and future research directions for real-time containers.

2 The Review Process

The systematic literature survey is carried out with the guidance in [17]. The research questions are defined together with search queries and sources of the studies and, subsequently, we extract the data and answer to the questions. We apply the snowballing [28] method to identify relevant papers outside the search query. Databases used: *Scopus* and *IEEE*. Only full peer review papers published between 2008-2019 are considered. We search the databases using the following search queries:

(Real-time OR RT) AND (Containers OR Container)

The search string extracts 1855 articles in *Scopus* and 609 articles in *IEEE*. Out of that, we identify 38 and 23 potentially relevant articles by the title. As the number of articles is low, we fully screen each potentially relevant paper (abstract and full text) to make the decision for inclusion/exclusion into this survey. In total, we include 14 papers as seen in Table 1.

2.1 Question Formalization

In this work, we elaborate the following questions:

- **RQ1:** *Why and in which context have real-time containers been used?* Answering this question will give an overview of the motivation behind the use of real-time containers, expected benefits and areas where real-time containers are used.
- **RQ2:** *What approaches are used for enabling real-time behavior of containers?* The answer will give an overview of the approaches and technologies, their combinations and their usages for the real-time container-based virtualization.
- **RQ3:** *What are the pitfalls and weak points of using real-time containers that prevent full adoption of such technology in industry?* The answer for this question will give a list of research challenges and problems for real-time container computing.

3 Container-based Virtualization

From the runtime perspective, a container is a set of resource-limited processes that are isolated from the rest of the system and from other containers. This is achieved by utilizing two Linux kernel features: (i) Namespaces and (ii) Control groups (cgroups). Namespaces virtualize global resources (e.g., processes, network, inter-process communication) in the way that a group of processes can see and use one set of resources while another group can use different set of resources. Cgroups provide a mechanism for aggregating and partitioning sets of tasks, and all their future children, into hierarchical groups with specialized behaviour [2]. It allows to organize processes hierarchically and distribute system resources along the hierarchy.

3.1 Container Platforms

There are several container solutions, all of them rely on cgroups and namespaces. Thus, all the platforms pose similar options and performance [23]. The philosophy of using the two most commonly used container platforms LXC and Docker differs. Docker containers are microservice-based (each container should contain a single application), whereas LXC, similarly to Virtual Machines, allows to run a complex ecosystem of applications which is beneficial for emulation of legacy systems.

3.2 Real-Time Containers

The term real-time implies that the correctness of the system depends not only on the results of the computation but also on the time at which the results are produced [8]. Real-time systems can be categorized into three groups: hard, firm and soft real-time. Missing a deadline in a hard real-time system may cause catastrophic consequences, whereas missing deadline in a firm real-time system leads to the complete loss of the utility of the result. Missing deadline in a soft real-time system just degrades the utility of the result.

A real-time container is a container that provides resource isolation, resource control and additionally provides time deterministic and predictable behavior for the containerized application.

3.3 Real-time Support of Linux

To ensure the time predictable behavior of the containers, the operating systems must provide such capability. Default (Vanilla) Linux does not give any time guarantees on execution of tasks and therefore the predictability is low [24]. However, there are several approaches to improve the predictability: the real-time patch that improves preemptability of the Linux kernel and co-kernel approaches that run a real-time micro kernel in parallel to the Linux kernel. Containerized applications are scheduled in the same way as native applications using the host's scheduler, the Default Linux kernel provides three schedulers: (i) *Completely Fair Scheduler (CFS)*: Aims to maximize CPU utilization while also maximizing interactive performance. It does not give any time guarantees. (ii) *Real-Time scheduler (RT)*: The scheduler allows to schedule tasks in the fixed priority manner using First In First Out or Round Robin policies. The tasks run till they yield or are preempted by higher priority tasks. The Real-Time group scheduling [3] extension allows to divide and allocate CPU time between real-time and non real-time tasks. (iii) *Earliest Deadline First Scheduler (EDF)* Uses Constant Bandwidth Server [6] and allows to associate to each task a budget and a period.

4 Survey Results

In this section, we summarise relevant papers. There are four main directions for enabling real-time behavior of containers: (i) real-time patch based, (ii) co-kernel based, (iii) hierarchical scheduling based and (iv) custom approach. A short summary is provided in Table: 1.

4.1 Methods Based on PREEMPT_RT Patch

The real-time patch (PREEMPT_RT) improves the kernel's locking primitives to maximize preemptible sections. The advantage of the patch is that there is no need for special libraries or API needed by the application developers.

Moga et al. [22] considers real-time containers in the context of industrial automation systems that works with real-time data and have real-time deadlines on detection and response to events. The paper emphasises the need for OS-level virtualization in an industrial automation and gives examples of timing requirements of industrial applications (e.g. motor drive typically requires cycle time between 1ms to 250 μ s) and a need for synchronization between the containers. The evaluation of the effects of containers on performance of industrial automation systems is provided in two cases: (i) Cyclic behavior of a containerized application, (ii) Virtual networking performance for communications between containers. Cyclic behavior test evaluates the ability to execute application logic at pre-defined intervals, measures accuracy and jitter. Virtual networking test evaluates the ability to communicate between co-located containers in a time-bounded manner. The researches see the real-time container computing as a promising technology, however communication mechanisms between containers are not clear.

The work in [16] (and previously [15]) addresses a container based architecture for real-time controllers that allow a flexible function deployment and a support of legacy control applications. Such architecture is needed to preserve a functionality of legacy control programs and to reduce maintenance cost of legacy systems (in which the software is often bounded to a specific hardware and software ecosystem). The researchers investigate the feasibility of building a real-time capable system (for legacy systems) based on real-time containers, they target PLCs and automation controllers with the cycle time between 100ms to 1s. They perform a set of tests under various load scenarios (i) using containerized applications inside of Docker and (ii) running complete operating system (PowerPC) inside LXC. They conclude that a containerized execution of control applications can meet requirements of PLCs and automation controllers.

From the latency point of view, Masek et al. [21] performed a literature review on sandboxed real-time software on the example of self-driving vehicles. The researchers were interested in the question: How does the execution environment influence the scheduling precision and input/output performance of a given application? The result shows that docker does not impose additional overhead (similarly to [19]) for scheduling and input/output performance. However, selecting the correct kernel has a greater impact on the scheduling precision and input/output performance of containers.

Mao et al. [20] uses real-time containers to enable software-based RAN (Radio Access Network) in order to avoid high capital and operating expenditures during deployment of new standards. However, the software based RAN has strict deadlines to satisfy (1ms). The researchers use real-time patch to decrease the latency, interestingly they improve the latency 13.9 times by applying the patch in comparison to the vanilla Kernel.

■ **Table 1** Summary of studies elaborating on real-time containers.

Study	Main focus	Approach & Technology	Communication aspects
Cinque et al. [9,10]	<ul style="list-style-type: none"> ■ Architecture definition ■ Faulty tasks monitoring ■ Implementation details 	<ul style="list-style-type: none"> ■ RTAI ■ Docker ■ Fixed priority scheduling 	–
Cucinotta et al. [5,11,12]	<ul style="list-style-type: none"> ■ Temporal Interference between containers 	<ul style="list-style-type: none"> ■ Hierarchical Scheduling 	–
Tasci et al. [26]	<ul style="list-style-type: none"> ■ Architecture definition ■ Real-time communication between containers 	<ul style="list-style-type: none"> ■ Combination of Real-Time patch and Xenomai ■ Docker 	Design of messaging system based on ZeroMQ.
Moga et al. [22]	<ul style="list-style-type: none"> ■ Feasibility study ■ Communication between containers ■ Communication overheads 	<ul style="list-style-type: none"> ■ Docker ■ Real-Time patch 	Network performance and overhead measurements between containers using default Docker Linux NAT Bridge.
Hofer et al. [18]	<ul style="list-style-type: none"> ■ Experimental comparison between Real-Time patch, Xenomai, Vanilla Linux 	<ul style="list-style-type: none"> ■ Real-Time patch ■ Xenomai ■ Vanilla Linux 	–
Goldschmidt et al. [15,16]	<ul style="list-style-type: none"> ■ Architecture definition ■ Feasibility study 	<ul style="list-style-type: none"> ■ Real-Time patch ■ Legacy systems emulation in real-time containers 	–
Telschig et al. [27]	<ul style="list-style-type: none"> ■ Model-based architecture and analysis ■ Dependable real-time container computing. 	<ul style="list-style-type: none"> ■ LXC 	–
Mao et al. [20]	<ul style="list-style-type: none"> ■ Minimizing latencies in software-based Radio Access Networks 	<ul style="list-style-type: none"> ■ Docker ■ Real-Time patch 	Application of fast packet processing using Intel Data Plane Development Kit.
Masek et al. [21]	<ul style="list-style-type: none"> ■ Systematic evaluation of sandboxed software 	<ul style="list-style-type: none"> ■ Real-Time patch 	–
Wu et al. [29]	<ul style="list-style-type: none"> ■ Dynamic CPU allocation for mixed-criticality real-time systems 	<ul style="list-style-type: none"> ■ Custom scheduling mechanism ■ Docker 	–

4.2 Methods based on Real-time Co-Kernel

In this approach, a real-time micro-kernel runs in parallel to Linux kernel. The real-time co-kernel handles time critical activities (e.g., handling interrupts and scheduling real-time threads), standard Linux kernel runs only when the co-kernel is idle. In comparison to the real-time patch, the co-kernel approach offers lower latencies and lower jitter. On the other hand, it requires special APIs, tools and libraries for the application development. Additionally, there are impediments with scaling co-kernel solutions on large platforms (e.g., many cores platforms). There are two co-kernel alternatives: Real Time Application Interface (RTAI) and Xenomai.

RTAI aims to minimize latencies to the lowest technically possible values. Real-time tasks are compiled as kernel modules and ran in the kernel space. Xenomai [14] is a fork of RTAI. Its mission is to enable real-time tasks in the user space. It consists of an emulation layer that is capable of reusing code from other RTOSes.

Tasci et al. [26] elaborates on modularization of real-time control applications into real-time containers. Such modular architecture needs two essential parts: (i) Computational part, enabled by a real-time operating system (combination of Xenomai and real-time patch), and (ii) Messaging part that allows passing messages between containers in a real-time manner. Traditional monolithic architectures communicate through function calls and shared memory, the containers do not make the assumption if they are running on the same host or in a distributed environment (they communicate through standard OS networking stack), therefore direct passing messages through shared memory is not directly supported. Hence, the researchers provide a design and implementation of a custom made real-time messaging system for containers based on ZeroMQ [4].

Hofer et al. [18] use the real-time containers in the context of control applications. The paper presents comparison between type 1 hypervisor, Vanilla Linux, Xenomai co-kernel and Linux with real-time patch for various idle and stress scenarios.

4.3 Method Based on Hierarchical Scheduling Of Containers

Inspired by a similar concept in the hypervisor-based virtualization where a global scheduler assigns CPU time for the virtual machines, the second layer scheduler schedules the individual tasks of the VM.

Cucinotta, Abeni et al. [5, 11, 12] proposed the use of real-time containers on the field of Network Function Virtualization (NFV), where the functionality of traditional physical network devices (e.g., firewalls) is transformed into software components (in containers) that are consolidated in a single computing device. NFV has critical latency requirements inducted by the need of time critical per-packet processing. The researchers modified the Linux scheduling mechanism to provide two levels hierarchical scheduling. First level Earliest Deadline First scheduler selects the container to be scheduled on each CPU. Subsequently the second level Fixed Priority scheduler selects a task in the container. CPU reservation (runtime quota and period) is assigned to each of the containers.

4.4 Custom Methods

Wu et al. [29] proposed the Flexible Deferrable Scheduler for containerized mixed-criticality real-time systems that consist of real-time and non real-time containers. The scheduler guarantees the allocated CPU capacity to real-time containers and dynamically distributes the unused capacity to non real-time containers. The work supplements Completely Fair Scheduler with a Workload Adjustment Module that collects CPU utilization by containers and Dynamic Adjustment Module that allocates CPU to the container.

Cinque et al. [10] (previously [9]) implemented real-time containers using Linux patched with real-time co-kernel (RTAI) and utilizing custom made monitoring and policy enforcing modules. Their solution allows to co-habit containers with different criticality levels and to prevent fixed-priority hard real-time periodic tasks inside of the containers to affect the temporal guarantees of other containers. The temporal guarantees are provided by two mechanisms: (i) proper tasks priority assignments to tasks inside the containers and (ii) monitoring and enforcing temporal protection policies. The former ensures that tasks inside of the high-criticality containers are assigned higher priorities than tasks in the lower-criticality containers and thus they are never preempted by tasks of lower criticality containers. The latter monitors the tasks and, in case of overruns or overtimes, it enforces one of the temporal protection policy (i.e., kill or suspend the faulty task, suspend the task until the next period).

5 Challenges of Real-time Container-based Virtualization

In the reviewed papers, we identified shortcomings and immature aspects of real-time container virtualization that prevents the expansion of the technology. Below, we listed them categorized in three groups: (i) tools support, (ii) real-time communication support, and (iii) miscellaneous.

Lack of tools for real-time container management. The reviewed papers emphasises a need for supporting tools for real-time containers. Tools that enable deployment on containers taking into account real-time requirements of containers and properties of computational nodes.

- The need for an orchestration tool that can schedule real-time containers based on pre-configured capabilities [18].
- Middleware that is aware of both communication needs as well as run-time and performance isolation needs [22].
- Framework to expose the runtime requirements of real-time application running inside containers and to enforce an optimal allocation of containers to resources [22].

Communication between real-time containers. Real-time communication between a container and its environment has to be further researched. Currently, the reviewed papers emphasize the following issues:

- Need for a real-time communication among containers [22].
- Further investigation on container security restricted container access and intra-container communication [18].
- A research on data management shared across containers [15].

Miscellaneous. In addition to generic issues that may harm the real-time behaviour (e.g., shared caches, memory and I/O), the studies reviewed highlight the following points and questions:

- Lack of safety, security analysis of real-time containers and vulnerability management for the acceptance in industry [15,27].
- Lack of latency and performance tests of recent releases of a patched Linux Kernel. As well as a proper analysis of configuration of the Linux kernel parameters that may improve overall task determinism. [18].

- The measurements of memory overhead of the container solution and is it acceptable for real world applications [15].
- Processes in different containers may use the same resources in the same way because of their independent views of the system (i.e., processes are not aware of a resource-limited isolated environment co-located with other containers). This results in poor resource utilization as well as a potential violation of the real-time execution assumptions [22].
- Container approaches are a new technology. Will this create problems due to its possible immaturity [15]?

6 Conclusion

Container-based virtualization has become popular as a lightweight alternative of hypervisor-based virtualization. The technology has proven its viability in large cloud-based systems, it has been adopted by a number of enterprise companies and it is supported by a large scale of tools (e.g., container orchestration and monitoring tools).

However, in industrial domains where the real-time behavior is required, the container-based virtualization seems not to be mature enough. In this paper, we summarize the research carried out in the field of real-time containers. We show in what contexts, what approaches and technologies are used, and what are the possible immaturity points of the real-time container-based virtualization.

References

- 1 Notes from a container. URL: <https://lwn.net/Articles/256389/>.
- 2 The Linux Kernel Archives. URL: <https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt>.
- 3 The Linux Kernel Archives. URL: <https://www.kernel.org/doc/Documentation/scheduler/sched-rt-group.txt>.
- 4 Zero MQ. URL: <https://zeromq.org/>.
- 5 Luca Abeni, Alessio Balsini, and Tommaso Cucinotta. Container-based real-time scheduling in the linux kernel. *SIGBED Rev.*, 2019.
- 6 Luca Abeni, Giuseppe Lipari, and Juri Lelli. Constant bandwidth server revisited. *Acm Sigbed Review*, 2015.
- 7 Thanh Bui. Analysis of docker security. *ArXiv*, abs/1501.02967, 2015. arXiv:1501.02967.
- 8 Giorgio C Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media, 2011.
- 9 Marcello Cinque and Domenico Cotroneo. Towards lightweight temporal and fault isolation in mixed-criticality systems with real-time containers. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2018.
- 10 Marcello Cinque, Raffaele Della Corte, Antonio Eliso, and Antonio Pecchia. Rt-cases: Container-based virtualization for temporally separated mixed-criticality task sets. In *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, 2019.
- 11 Tommaso Cucinotta, Luca Abeni, Mauro Marinoni, Alessio Balsini, and Carlo Vitucci. Virtual network functions as real-time containers in private clouds. In *IEEE CLOUD*, 2018.
- 12 Tommaso Cucinotta, Luca Abeni, Mauro Marinoni, Alessio Balsini, and Carlo Vitucci. Reducing temporal interference in private clouds through real-time containers. In *2019 IEEE International Conference on Edge Computing (EDGE)*, pages 124–131, 2019. doi: 10.1109/EDGE.2019.00036.

- 13 W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2015.
- 14 Philippe Gerum. Xenomai-implementing a rtos emulation framework on gnu/linux. *White Paper, Xenomai*, pages 1–12, 2004.
- 15 Thomas Goldschmidt and Stefan Hauck-Stattelmann. Software containers for industrial control. In *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2016.
- 16 Thomas Goldschmidt, Stefan Hauck-Stattelmann, Somayeh Malakuti, and Sten Grüner. Container-based architecture for flexible industrial control applications. *Journal of Systems Architecture*, 84:28 – 36, 2018. URL: <http://www.sciencedirect.com/science/article/pii/S1383762117304988>, doi:<https://doi.org/10.1016/j.sysarc.2018.03.002>.
- 17 Jo Hannay, Dag Sjøberg, and Tore Dybå. A systematic review of theory use in software engineering experiments. *Software Engineering, IEEE Transactions on*, 2007.
- 18 Florian Hofer, Martin Sehr, Antonio Iannopollo, Ines Ugalde, Alberto Sangiovanni-Vincentelli, and Barbara Russo. Industrial control via application containers: Migrating from bare-metal to iaas. In *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 62–69, 2019. doi:10.1109/CloudCom.2019.00021.
- 19 A. Krylovskiy. Internet of things gateways meet linux containers: Performance evaluation and discussion. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015.
- 20 C. Mao, M. Huang, S. Padhy, S. Wang, W. Chung, Y. Chung, and C. Hsu. Minimizing latency of real-time container cloud for software radio access networks. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, 2015.
- 21 Philip Masek, Magnus Thulin, Hugo Andrade, Christian Berger, and Ola Benderius. Systematic evaluation of sandboxed software deployment for real-time software on the example of a self-driving heavy vehicle. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, 2016.
- 22 Alexandru Moga, Thanikesavan Sivanthi, and Carsten Franke. Os-level virtualization for industrial automation systems: are we there yet? In *SAC '16*, 2016.
- 23 Roberto Morabito, Jimmy Kjällman, and Miika Komu. Hypervisors vs. lightweight virtualization: a performance comparison. In *2015 IEEE International Conference on Cloud Engineering*, 2015.
- 24 Claudio Scordino and Giuseppe Lipari. Linux and real-time: Current approaches and future opportunities. In *IEEE International Congress ANIPLA*, 2006.
- 25 Cristian Spoiala, Alin Calinciuc, Cornel Turcu, and Constantin Filote. Performance comparison of a webrtc server on docker versus virtual machine. *13th International Conference on DEVELOPMENT AND APPLICATION SYSTEMS, Suceava, Romania, May 19-21, 2016*, 2016.
- 26 Timur Tasci, Jan Melcher, and Alexander Verl. A container-based architecture for real-time control applications. In *2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. IEEE, 2018.
- 27 Kilian Telschig, Andreas Schonberger, and Alexander Knapp. A real-time container architecture for dependable distributed embedded applications. *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 1367–1374, 2018.
- 28 Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14*, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2601248.2601268.
- 29 J. Wu and T. Yang. Dynamic cpu allocation for docker containerized mixed-criticality real-time systems. In *2018 IEEE International Conference on Applied System Invention (ICASI)*, 2018.

Evaluation of Burst Failure Robustness of Control Systems in the Fog

Nils Vreman

Department of Automatic Control, Lund University, Sweden
nils.vreman@control.lth.se

Claudio Mandrioli

Department of Automatic Control, Lund University, Sweden
claudio.mandrioli@control.lth.se

Abstract

This paper investigates the robustness of control systems when a controller is run in a Fog environment. Control systems in the Fog are introduced and a discussion regarding relevant faults is presented. A preliminary investigation of the robustness properties of a MinSeg case study is presented and commented. The discussion is then used to outline future lines of research.

2012 ACM Subject Classification Computer systems organization → Sensor networks

Keywords and phrases Networked Control Systems, Stability Analysis, Control over Internet, Fault Tolerance

Digital Object Identifier 10.4230/OASICS.Fog-IoT.2020.8

Funding This work was supported by the ELLIIT Strategic Research Area; by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation; and by the ADMORPH project.

Nils Vreman: ELLIIT, ADMORF

Claudio Mandrioli: WASP

Acknowledgements The authors would like to thank their supervisors Anton Cervin and Martina Maggio for the interesting insights given during the writing of this paper.

1 Introduction

In recent years there has been a trend to move towards decentralized and distributed software infrastructure in industry [1]. This interest rises from the advantages of decoupling the control of the plant from the physical location and allowing for easier software update patches, lower maintenance costs, easier integration and optimization of individual components, among multiple others. This shift in paradigms is based on the emerging 5G-network with low latency, higher bandwidth, lower cost, and more reliable communication channels [4, 9]. However, neither the 5G-network nor the Fog that follows, are exempt from faults [5, 12]. The wireless and distributed nature of the Fog introduces different faults from the ones addressed in classical control theory [13].

In real-time control systems, where reliability and timing constraints are of utmost importance, the faults need to be analyzed thoroughly [10]. Recently, [11] showed that it is possible for a control system to run in a Fog setting. The authors managed to control a plant in real-time whilst migrating a controller from the near vicinity of the plant to two datacenters, located at vastly different places. This result shows that it is possible to use the Fog as a platform for distributed control but at the same time poses questions on its general limitations and applicability. There is therefore need to develop methodologies for the study of reliability and robustness of a control system in the Fog environment.



© Nils Vreman and Claudio Mandrioli;
licensed under Creative Commons License CC-BY
2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020).

Editors: Anton Cervin and Yang Yang; Article No. 8; pp. 8:1–8:8

OpenAccess Series in Informatics



OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The general set-up, where controller and plant are connected through a network, has been studied under the name of networked control systems [7]. We propose an extension of the work on networked control systems based on types and patterns of faults, characteristic to the Fog environment. In this environment, faults can happen in the computation, actuation, or sensing nodes as well as in the transmission between these nodes. Faults may appear in the computation node due to overloads generated by other running tasks. Communication faults might instead appear due to signal disturbances or channel overloads created by other IoT-devices. In both cases the faults are likely to appear in **bursts**, meaning that the fault appears at an arbitrary point in time and persists for some time before disappearing.

Control systems seem to guarantee an intrinsic robustness toward these faults but its boundaries are yet to be explored. We perform a preliminary investigation of the topic by discussing the possible faults as well as simulating their possible outcomes. Namely, the main contributions of this paper are:

- (i) a discussion on the specific faults introduced by a distributed Fog environment,
- (ii) an investigation on the effect of bursts of faults on a control system,
- (iii) a simulation based stability region for a particular case study.

The rest of the paper is structured as follows: Section 2 introduces control systems in the Fog and discusses the potential faults that can appear. Section 3 presents and discusses the case study of a MinSeg. Finally Section 4 summarizes the paper and outlines future research directions.

2 System Model

2.1 Control Systems

The purpose of a control system is to make a process behave according to some given requirements. Control systems are generally composed of a physical process and a controller. The controller is implemented as a software that receives measurements from the process, elaborates them, and decides accordingly how to steer the actuators of the process. Within this work we study the specific (but still common) case in which the controller implementation is split in two different components: a state observer and a control law. This is often needed due to the states of the system differing from what we can measure. A representation of these components and their interaction is shown in the block diagram of Figure 1.

For capturing the behaviour of the process, the most common class of models used are the so-called time-invariant *state-space models* [2]. State-space models take the form

$$\begin{aligned} \dot{x} &= f(x, u), \\ y &= g(x, u) \end{aligned} \tag{1}$$

where u is a vector of actuation variables decided by the control law (also called the *inputs* of the process), y is a vector of measured variables (also called the *outputs* of the process), and x is a vector of *states* of the process. Therefore, the first equation describes how the system evolves given the current state and input, whilst the second equation instead describes how the measurements are connected to the actual state of the system.

The purpose of the observer is to estimate x given the known input u and measured output y . The most common state observer is called the Luenberger observer [2]. This observer is based on running a simulation of the process according to Equation (1) and correcting it given the discrepancy between the expected measurement from the simulation

and the true measurement. In this way, the information contained in the model is merged with the measurements of the process. By calling the estimated state \hat{x} and expected output \hat{y} the Luenberger observer is implemented with the following equation:

$$\begin{aligned}\hat{y} &= g(\hat{x}, u), \\ \hat{\dot{x}} &= f(\hat{x}, u) + K(y - \hat{y})\end{aligned}\tag{2}$$

where the constant K (also called the *observer gain*) can be chosen using different techniques (e.g. pole placement or Kalman filtering) [2]. Intuitively a high value of K represents that the observer will rely more on the measurements and a low value will make the observer rely more on the simulated model of the process.

The estimated state \hat{x} is then used by the controller, together with the desired behaviour r to compute the control action u . The most common class of controllers are linear controllers which can be written as

$$u = L(r - \hat{x}),\tag{3}$$

where L is called the *control gain* and can be designed using standard techniques from control theory – e.g. pole-placement or LQG control [2].

The mentioned design methodologies provide formal guarantees on the stability and performance of the control system. Stability is the most fundamental property required in a control system and states that the system variables x , \hat{x} , u , and y over time will not diverge but instead converge to some finite value. Intuitively these guarantees depend on the accuracy of the available model. In a control system, the capability of satisfying the requirements in presence of modelling errors, disturbances, and component faults is called **robustness**. In this work we analyze the robustness to the Fog faults in terms of stability guarantees. Specifically, in the Fog context, we want to evaluate how tolerant a control system is to faults.

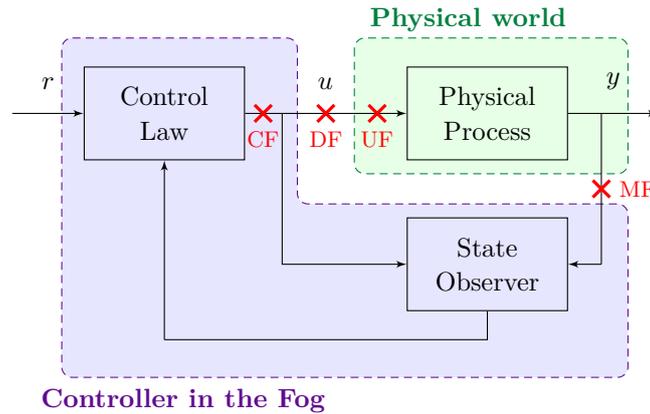
2.2 Control Systems in the Fog

In traditional control systems, the control software is executed by an embedded device that is attached to the physical process. Leveraging the emerging Fog network, it becomes natural to move the controller into the Fog. This allows the control system to access higher computational power, ease software updates, synchronize with other IoT-devices as well as alter the hardware during runtime.

When the controller is run in the Fog, it interacts with the plant through wireless channels, whilst sharing the computational power with other processes. Therefore, both the communication channel and the computation platform are subject to the interference of other IoT-devices. These phenomena introduce new and specific disturbances different from the ones seen in traditional control systems. Evaluating the performance of control systems in their presence is critical for the safe and optimal implementation of a control system in the Fog.

In Figure 1, where the block diagram of the control system is shown, the dashed boxes highlight that the controller (both state observer and control law) is executed in the Fog and that the process is placed in a different physical location. This shows that the desired behaviour r , the control actuation u , and the measurement y are the signals traveling on wireless communication channels.¹

¹ Different set-ups could be considered, for example the state observer and the control law could be run in different IoT nodes. In this work we limit our study to this case being it the closest to the traditional set-up.



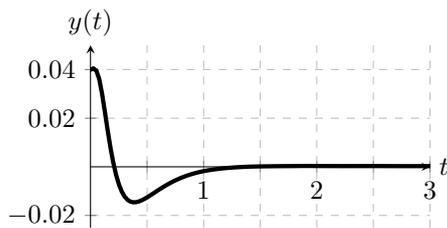
■ **Figure 1** A control system in the Fog context. The purple dashed block illustrates all the components located in the Fog, whilst the green dashed block represents the physical process we are trying to control. Transmission channels are represented by black arrows and red crosses are used to indicate a transmission fault.

2.3 Fog Faults and Control Systems

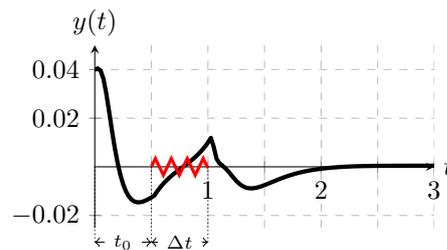
Given the Fog-IoT set-up described in the previous section, faults could be introduced when communicating the signals r , u , and y as well as when executing the controller. Among these we exclude faults in the communication of r , since this signal is set by an external operator and not affected by the control system.

In this work we discuss the following scenarios (shown in Figure 1 as red-crosses in the control system block diagram):

- **Computation Faults [CF]:** the IoT computation node does not manage to execute the controller within the real-time constraints of the control system. Therefore, the actuator does not receive any actuation command and the controller does not update. This could happen, for example, if the node was suddenly overloaded by other computations.
- **Detected Actuation Transmission Faults [DF]:** the communication channel fails in transmitting the actuation signal u but the control software detects it and can take counteraction. The actuator does not receive any command and the controller updates accordingly. This could for example happen if the communication channel is kept busy by other IoT devices.
- **Undetected Actuation Transmission Faults [UF]:** the communication channel fails in transmitting the actuation signal u and the control software is not aware of this and can therefore not take any counteraction. The actuator does not receive any command and the controller updates normally. This could happen, for example, if the controller get access to the communication channel but the channel itself does not succeed in transmitting the signal.
- **Measurement Transmission Faults [MF]:** the communication channel fails in transmitting the measurement signal y . The control software does not receive it and can therefore take counteraction. This could for example happen if the communication channel is either busy or simply fails to transmit the signal.



■ **Figure 2** Time series of the MinSeg leaning angle under regular conditions.



■ **Figure 3** Time series of the MinSeg leaning angle in the presence of a burst of computation faults starting at $t_0 = 0.5$ with a duration of $\Delta t = 0.5$.

2.4 Fault Patterns

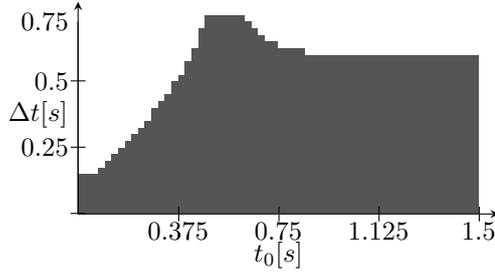
In this work we consider fault patterns of the **burst** type, meaning that a given fault happens sporadically and continuously for a relatively short time. The reason for considering this type of faults is that Fog-IoT components are expected to incur temporary overload periods that will result in temporary unavailability.

We define burst faults as a sequence of consecutive component faults that happens at an arbitrary point in time and does not repeat. Therefore, a burst fault of a given type is defined by two quantities: an initial time t_0 and a duration Δt . A graphical representation of a burst fault and the corresponding process is shown in Figure 3, as opposed to the system response in regular conditions shown in Figure 2. In Figure 3, the system experiences a burst of computation faults at time $t_0 = 0.5$ with length $\Delta t = 0.5$. The red zig-zag line (in the x-axis) shows the interval in which the fault occurs and the plot shows the time series of the leaning angle of the MinSeg.

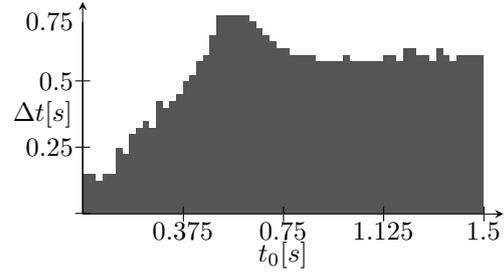
When the actuator does not receive an updated control signal, we implemented a *zeroing* strategy [8]. This means that the variable u is set to zero in the presence of computation faults (CF), detected (DF), and undetected actuation transmission faults (UF). We chose a zeroing strategy over the alternative of re-applying the previous control signal. In fact, in many control systems, after the computation of the control signal, additional steps are performed, for example to transform the control signal between different coordinate spaces (e.g., dynamic coordinate systems). The presence of faults would render this additional computation (and holding the previously computed signal) infeasible, therefore justifying our choice.

3 Results and Discussion

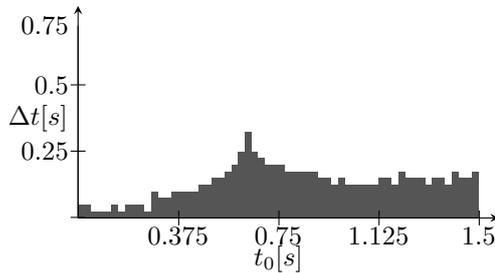
We will in this section evaluate a model of a real control system through simulations. From the simulation results, we discuss the system robustness to the transmission disturbances presented in Section 2. The real-world process that we chose to analyze is a MinSeg [6] controlled via bluetooth technology. Due to the fact that the MinSeg is inherently unstable, with fast dynamics, it is a relevant process for our experiments. The performance degradation of the control system is therefore clearly exposed when computation and transmission errors are introduced.



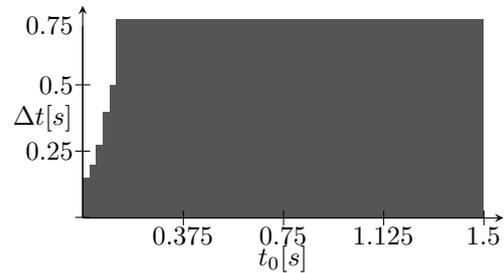
■ **Figure 4** Stability region for simulations using computation faults [CF].



■ **Figure 5** Stability region for simulations using detected actuation transmission faults [DF].



■ **Figure 6** Stability region for simulations using undetected actuation transmission faults [UF].



■ **Figure 7** Stability region for simulations using measurement transmission faults [MF].

We ran the simulations using Matlab². In particular we simulated the timing interactions between the network components and the control system using the research tool called *TrueTime* [3]³. TrueTime is a Matlab multi-purpose toolbox primarily used for analyzing the complex timing properties of real-time, networked control systems.

We injected the burst disturbances (as described in Section 2.4) into the control system during a transient. A transient is when the process moves from an arbitrary state to an equilibrium. An equilibrium is a state from which the system will not diverge unless in presence of some external input or disturbance. This is a standard, however not exhaustive, way to evaluate the performance of a control system.

We ran simulations for each type of fault with one burst each. We considered bursts of different length Δt , starting at different points t_0 of the transient, as discussed in Section 2.4. The stability results from the combinations of t_0 and Δt values are then plotted together. We define a simulation stable when the angle of the MinSeg does not exceed a given threshold α for the entire duration of the simulation. Consequently, when the threshold is exceeded, the simulation is marked as unstable. In fact, exceeding this threshold implies that the angle of the process is too large for the control system to be able to bring the MinSeg back to an upright position. Figures 4, 5, 6, and 7 show the stability regions for each of the faults defined in Section 2.3 given different configurations of Δt and t_0 . Each point (x, y) in the plot represents a simulation using a specific $(\Delta t, t_0)$ combination. Unstable simulations are marked with white and stable simulations are marked with gray.

² <https://se.mathworks.com/products/matlab.html>

³ <http://www.control.lth.se/research/tools-and-software/truetime/>

As we can see in Figure 2, for values of $t_0 < 0.75s$, the burst happens while the system is still in a transient state. Conversely, for values of $t_0 > 0.75s$ the system has reached its equilibrium, and the simulations are not changing anymore. For all the faults, this provides the intuition that in an equilibrium state the robustness to burst for a control system could be quantified. Still, there are differences between the specific faults.

Figure 5 shows the results of the simulations in presence of detectable actuation transmission faults. The behaviour of the system appears similar to the case of computation faults (Figure 4). Intuitively, in both cases, the system is not actuated. However, the difference between the two faults is the fact that the state observer is not updated during the computation faults burst. According to the simulations, the difference does not affect the robustness of the system. In the future we plan to investigate the generality of this finding.

Figure 6 shows the results of the simulations in presence of undetectable actuation transmission faults. The system shows less robustness with respect to the previous two cases. This can be attributed to the state observer being updated with the wrong control signal and therefore diverging from the actual state of the system. Despite this, the general behaviour of the stability region shows a similar trend to the two faults considered previously. In general, from the comparison of Figure 5 and 6, undetectable actuation transmission faults can be more harmful than detectable ones. This should be taken into account when the Fog infrastructure is implemented, e.g. by implementing detection algorithms for transmission faults.

Figure 7 shows the results of the simulations in presence of measurement transmission fault. Most of the simulations expose a stable behaviour. This is due to the perfect coherence between the model used in the observer and the model used to simulate the process. Since the two models are the same, despite the lack of feedback, the estimated states do not diverge from the actual states of the process. This will not be true for a real implementation of the system due to modeling errors and disturbances. The unstable simulations, that appear for small values of t_0 , are due to the observer not having enough time to start tracking the states of the process.

4 Conclusions

This paper presents preliminary work investigating the robustness to computation and communication faults of control systems in the Fog. The simulations show that a control system has an intrinsic robustness to the faults characteristic of this environment. Further investigations should be based on a formal analysis of the system properties. The relevance of the state observer, in the presence of the discussed faults, has been emphasized through simulations. In future work we plan to evaluate solutions that handle burst faults.

References

- 1 M. Aazam, S. Zeadally, and K. A. Harras. Deploying fog computing in industrial internet of things and industry 4.0. *IEEE Transactions on Industrial Informatics*, 14(10):4674–4682, October 2018. doi:10.1109/TII.2018.2855198.
- 2 Karl Johan Åström and Richard M. Murray. *Feedback Systems : An Introduction for Scientists and Engineers*. Princeton University Press, 2008.
- 3 A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K. E. Årzén. How does control timing affect performance? analysis and simulation of timing using jitterbug and truetime. *IEEE Control Systems Magazine*, 23(3):16–30, 2003. doi:10.1109/MCS.2003.1200240.

- 4 Tommaso Cucinotta, Mauro Marinoni, Alessandra Melani, Andrea Parri, and Carlo Vitucci. Temporal isolation among lte/5g network functions by real-time scheduling. In *Proceedings of the 7th International Conference on Cloud Computing and Services Science*, CLOSER 2017, page 368–375, Setubal, PRT, 2017. SCITEPRESS - Science and Technology Publications, Lda. doi:10.5220/0006246703680375.
- 5 F. Foukalas, P. Pop, F. Theoleyre, C. A. Boano, and C. Buratti. Dependable wireless industrial iot networks: Recent advances and open challenges. In *2019 IEEE European Test Symposium (ETS)*, pages 1–10, May 2019. doi:10.1109/ETS.2019.8791551.
- 6 B. Howard and L. Bushnell. Enhancing linear system theory curriculum with an inverted pendulum robot. In *2015 American Control Conference (ACC)*, pages 2185–2192, July 2015. doi:10.1109/ACC.2015.7171057.
- 7 Dimitrios Hristu-Varsakelis, William S. Levine, R. Alur, K.-E. Arzen, John Baillieul, and T. A. Henzinger. *Handbook of Networked and Embedded Control Systems (Control Engineering)*. Birkhauser, 2005.
- 8 S. Linsenmayer and F. Allgower. Stabilization of networked control systems with weakly hard real-time dropout description. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 4765–4770, December 2017. doi:10.1109/CDC.2017.8264364.
- 9 Daniel Jun Xian Ng, Arvind Easwaran, and Sidharta Andalam. Contract-based hierarchical resilience framework for cyber-physical systems: Demo abstract. In *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS '19*, page 324–325, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3302509.3313323.
- 10 M. Pohjola, S. Nethi, and R. Jantti. Wireless control of mobile robot squad with link failure. In *2008 6th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks and Workshops*, pages 648–656, April 2008. doi:10.1109/WIOPT.2008.4586154.
- 11 Per Skarin, Karl-Erik Årzén, Johan Eker, and Maria Kihl. Cloud-assisted model predictive control. In *2019 IEEE International Conference on Edge Computing*, pages 110–112. IEEE - Institute of Electrical and Electronics Engineers Inc., August 2019. doi:10.1109/EDGE.2019.00033.
- 12 W. Wang, D. Mosse, and A. V. Papadopoulos. Packet priority assignment for wireless control systems of multiple physical systems. In *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, pages 143–150, May 2019. doi:10.1109/ISORC.2019.00036.
- 13 L. Zhang, L. Zhao, and L. Li. Sliding mode control for network control systems with packets dropout. In *2015 34th Chinese Control Conference (CCC)*, pages 6592–6596, July 2015. doi:10.1109/ChiCC.2015.7260677.

Next-Generation SDN and Fog Computing: A New Paradigm for SDN-Based Edge Computing

Eder Ollora Zaballa 

DTU Fotonik, Technical University of Denmark, Lyngby, Denmark
eoza@fotonik.dtu.dk

David Franco 

Department of Communications Engineering, University of the Basque Country UPV/EHU,
Bilbao, Spain
david.franco@ehu.eus

Marina Aguado 

Department of Communications Engineering, University of the Basque Country UPV/EHU,
Bilbao, Spain
marina.aguado@ehu.eus

Michael Stübert Berger 

DTU Fotonik, Technical University of Denmark, Lyngby, Denmark
msbe@fotonik.dtu.dk

Abstract

In the last few years, we have been able to see how terms like Mobile Edge Computing, Cloudlets, and Fog computing have arisen as concepts that reach a level of popularity to express computing towards network Edge. Shifting some processing tasks from the Cloud to the Edge brings challenges to the table that might have been non-considered before in next-generation Software-Defined Networking (SDN). Efficient routing mechanisms, Edge Computing, and SDN applications are challenging to deploy as controllers are expected to have different distributions. In particular, with the advances of SDN and the P4 language, there are new opportunities and challenges that next-generation SDN has for Fog computing. The development of new pipelines along with the progress regarding control-to-data plane programming protocols can also promote data and control plane function offloading. We propose a new mechanism of deploying SDN control planes both locally and remotely to attend different challenges. We encourage researchers to develop new ways to functionally deploying Fog and Cloud control planes that let cross-layer planes interact by deploying specific control and data plane applications. With our proposal, the control and data plane distribution can provide a lower response time for locally deployed applications (local control plane). Besides, it can still be beneficial for a centralized and remotely placed control plane, for applications such as path computation within the same network and between separated networks (remote control plane).

2012 ACM Subject Classification Networks → Programmable networks

Keywords and phrases SDN, P4, P4Runtime, control planes, Fog, Edge

Digital Object Identifier 10.4230/OASICS.Fog-IoT.2020.9

Funding This research has been supported by the Spanish Ministry of Science, Innovation, and Universities within the project TEC2017-87061-C3-2-R (CIENCIA/AEI/FEDER, UE).

David Franco: Supported by aforementioned project.

Marina Aguado: Supported by aforementioned project.



© Eder Ollora Zaballa, David Franco, Marina Aguado, and Michael Stübert Berger;
licensed under Creative Commons License CC-BY

2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020).

Editors: Anton Cervin and Yang Yang; Article No. 9; pp. 9:1–9:8

OpenAccess Series in Informatics



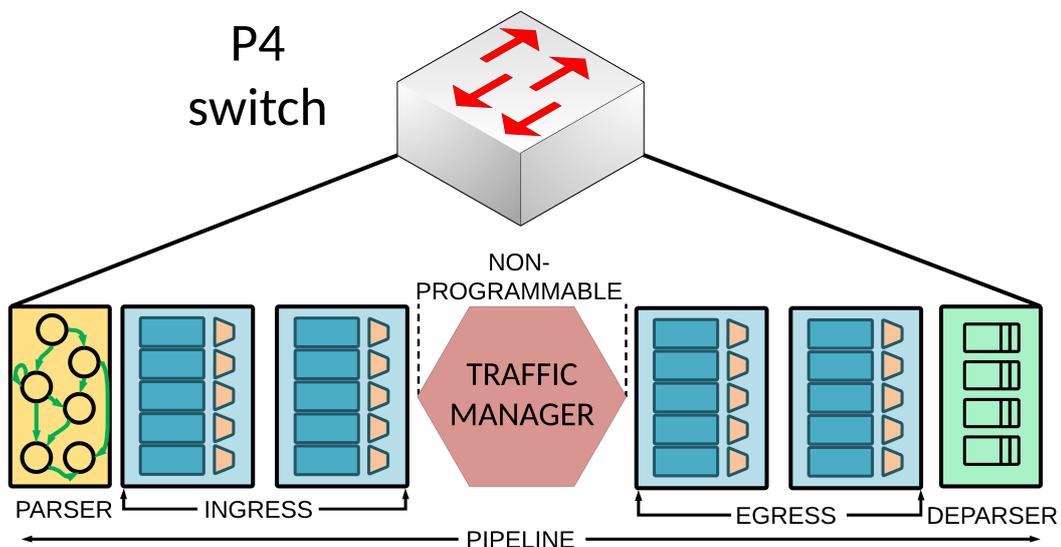
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The concept of Cloud Computing has turned into a revolution for computer science engineering, becoming essential when designing modern communication networks. With the advances in computational architectures and distributed computing, other approaches like Mobile Edge Computing, Fog Computing, and Cloudlet have arisen as relevant terms to describe new data processing procedures that bring computation closer to the clients.

The next-generation requirements created by offloading processing to the Edge need a way to be addressed. In recent years, Software-Defined Networking (SDN) has seen an evolution to develop full network programmability. The requirements imposed by Edge-based data processing can be addressed by SDN. We can see in Figure 2 how SDN networks have evolved, starting from traditional forwarding devices, to OpenFlow-based network and then to next-generation SDN networks based on P4 (data plane programming language) [2] [3] and P4Runtime (runtime protocol to control P4-defined switches) [4].

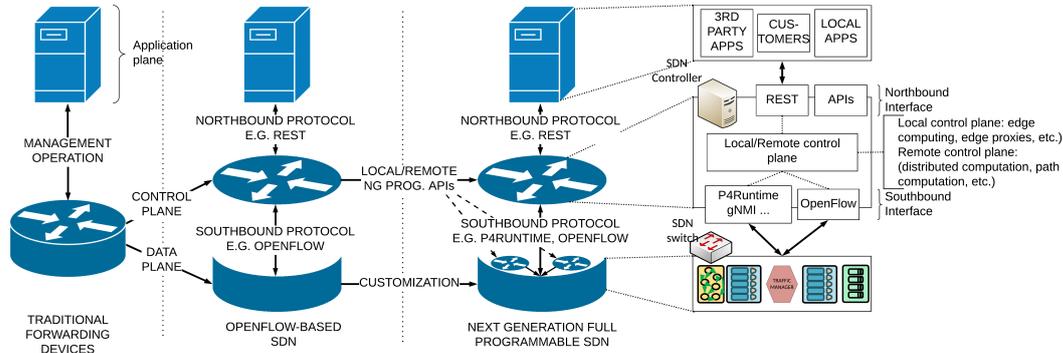
In particular, P4 is a language to define the data plane behavior for forwarding devices. As we can see in Figure 1, we can define how packets are parsed (custom headers), how packets are treated (custom tables and actions at ingress or egress stages of a pipeline) and how packets are sent again into the network (custom deparser, add or remove headers). We can also define how to count for packets/bytes, define network meters or be able to program our third-party external functions (also known as *externs*) that can be integrated into the data plane. Depending on the P4 programmable hardware, packets can also be treated differently in queues, being able to assign a particular priority.



■ **Figure 1** P4 switch pipeline definition by V1model architecture.

Thanks to the advent of data plane programming in the last few years, we can extend the functionality of networking devices. For instance, they are capable of performing computational operations in the data plane, at line speed. This concept is known as in-network computing, and it has several classes of applications such as network functions, caching or data aggregation. This computational approach yields new control-to-data plane protocols (e.g. P4Runtime), which opens new opportunities to define the control plane of next-generation SDN networks. Fog networks can especially benefit from these as control

planes can work both locally and remotely at the same time for the same data planes. Therefore, this paper describes several novel approaches to locally and remotely deploy SDN control to benefit Fog computing networks. For instance, a local control plane can address requests that require quick responses (e.g. Edge Computation), while a remote controller is responsible for making decisions that demand a global view of the network (e.g. path computation).



■ **Figure 2** Evolution of SDN from traditional and unified control and data plane devices towards first generation OpenFlow-enabled SDN and current full programmable control and data planes.

Apart from the details we have offered about SDN and P4, we offer further details about the related work in Section II. We also provide further details on how exactly P4 and P4Runtime can be beneficial for Fog computing in Section III. Section IV describes a novel paradigm in which SDN network devices can be used for edge computing, specifically accounting for distributed control planes that are integrated in a cross-layer way. Finally, Section V offers a conclusion for the content explained in this paper.

2 Related Work

In this section, we will try to offer a variety of work-related to SDN and Fog computing. We have tried to focus on different use cases using SDN although most of the work done in the past years focuses on surveys that relate SDN to Edge/Fog computing [1] [10] [8]. There is not as much work in the practical and implementation side done with SDN and Edge/Fog compared to surveys, however, this section addresses a variety of topics inside Edge/Fog in which SDN has been interesting for.

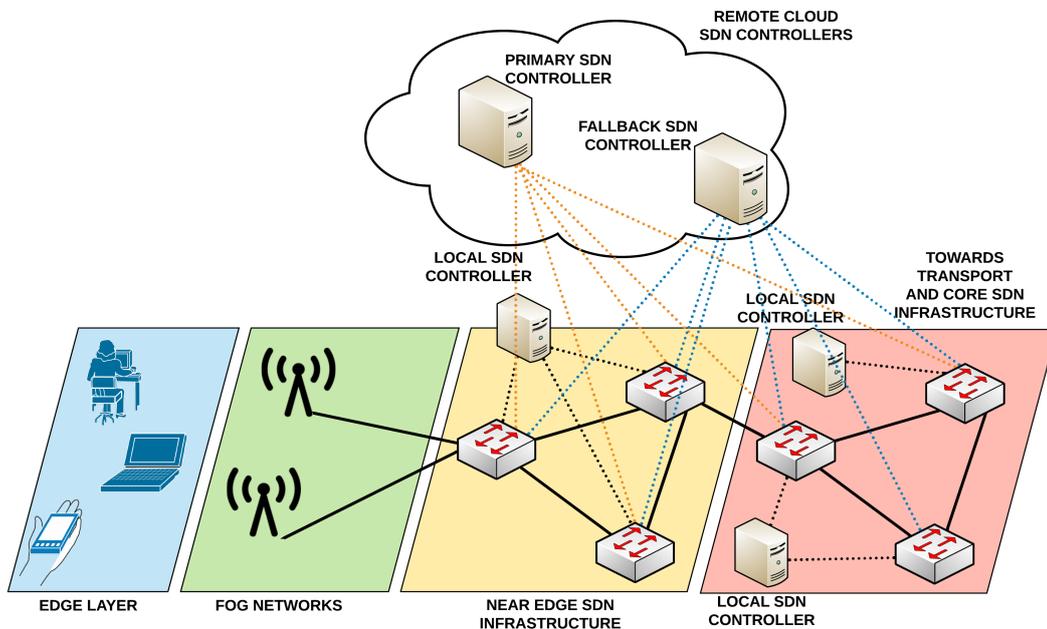
The work we followed in [1] offers a wide variety of use cases for SDN within edge computing. The authors mention several of the key areas where SDN can benefit Edge networks using some of the key papers of the work. As they only briefly mention P4, we can demonstrate that the main areas of the paper have not deeply considered P4 or P4Runtime. However, some of the main areas are Service-Centric Implementation, Adaptability, Interoperability, High Resolution, and Effective control, etc. The role of SDN in these areas encompasses several control plane applications support, runtime service reconfiguration or mobility management support. The authors still point out that future research is going in the direction of improving network virtualization, enhancing north and southbound protocols to support future services and improve scalability and reliability.

Authors in [10] also briefly go through some of the key areas for SDN controllers in Fog computing. Authors mention that controllers should perform actions within data traffic management, resilience, and Fog orchestration. Time sensitivity is also mentioned as a key area for SDN controllers to manage when dealing proactively or reactively with table entries.

As mentioned in [6], authors show how to integrate SDN and Cloud-RAN within 5G to globally allocate resources for VANETs. The goal for SDN within the Fog network remains unclear but authors seem to integrate SDN controllers hierarchically on top of Fog computing BBU controllers to manage Fog orchestration and resources apart from doing regular network management tasks.

Specifically focusing on P4, authors in [9] explain how to integrate P4 into a multi-layer edge scenario. The authors propose 3 cases in which dynamic Traffic Engineering (TE) is covered and also cybersecurity which is addressed through P4-based solutions. The authors test their ideas in virtual switches like BMv2 and physical switches like the NetFGPA. Their tests report successful dynamic TE and cybersecurity mitigation without controller intervention. However, the authors still do not look into the same topics as we do in this paper for cross-layer multi-SDN control planes.

3 P4 and P4Runtime as a tool for fog computing



■ **Figure 3** Local and remote control planes managing the data plane from Fog network and Core network based programmable data planes.

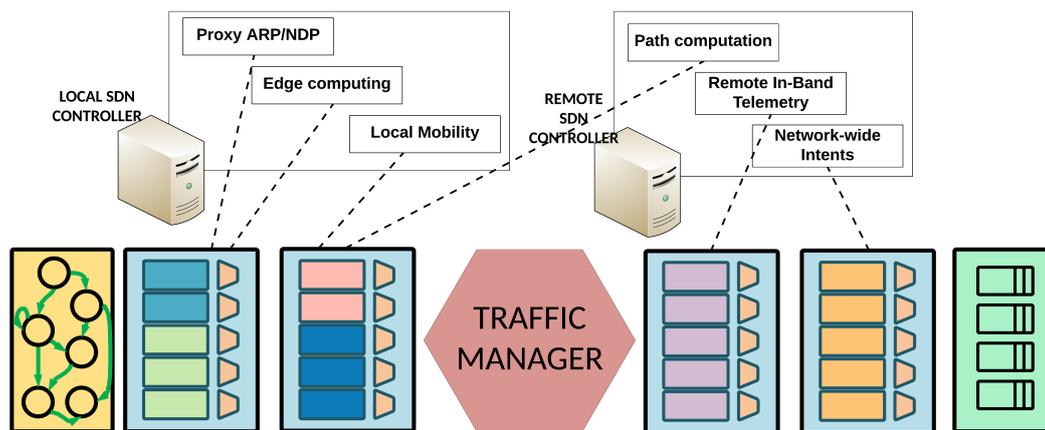
In 2008 an article describing a protocol called OpenFlow (OF) [7] gained enormous traction. The authors describe a protocol to encourage self-programmed control planes to program switches' forwarding tables. Switches would support a set of protocols (Ethernet, IPv4, TCP, etc.) and forward traffic depending on rules installed by SDN controllers. The supported matching engine has evolved to support new header fields (i.e. more protocols) as OpenFlow versions got developed. For instance, the OpenFlow version 1.5.1 supports over 41 fields. However, the researchers decided to develop new data and control plane programmability approaches. In 2014, the P4 language emerged to overcome some of the limitations that OpenFlow could have imposed. P4 encourages protocol evolution, enables faster design and better development cycle and provides new data plane monitoring techniques.

With P4, developers can define which headers a switch can understand, how to parse packets and customize matching criteria and actions. Moreover, as P4 is a programming language to describe the behaviour of the data plane, the entire processing takes place in the ASIC.

The limitations found in legacy SDN do not only refer to the data plane. The use of OpenFlow also imposes specific architectures that controllers need to follow. For instance, two different controllers can not control the same OpenFlow data plane, unless a layer between control and data plane is included [11]. Therefore, the support for multiple controllers over the same pipeline is not natively included in OpenFlow while it is in P4Runtime. This is further explained in the next sections.

While Fog computing has not been the main field for SDN study we believe that P4 and P4Runtime have strong arguments to be included at the Edge too. This section aims to strengthen the use of P4 within 5 main areas that make it relevant to be considered as the driver for Edge/Fog computing:

- Customized protocols: Within the core use case of P4 it is the definition of standardized and custom protocols. The definition of the headers, tables, and actions in P4 provides the means to build custom protocols.
- Protocol translation: As a consequence of being able to program which headers are parsed and which actions are executed, the pipeline can also enable and disable headers on demand (e.g. Tunneling packets between from Fog/Edge environments and also towards core).
- Monitoring: Having a custom data plane enables new telemetry protocols (e.g. In-band telemetry (INT)) to monitor the behavior and performance of packets within P4-programmable devices. Besides being able to create, modify and forward telemetry packets, the devices need to be able to monitor the data plane at the edge. To support this use case, data planes generally offer information via metadata available while packets traverse the data plane. This can be a beneficial use case to monitor packets at the edge, exporting information to network managers.
- Data plane partition: Being able to create tables (define maximum entry size, keys to match, actions and parameters, etc.) also brings new ways to distribute data plane functionality. Custom data planes can bring new ways of dealing with packets within the pipeline. For instance, one table can be used to learn new hosts and prevent spoofing while other tables can perform L2/L3 forwarding. This concept is tightly related to *controller partition*, explained in the next section.
- Edge cache: One of the areas with the biggest potential is using programmable switches as network caches. This functionality has already been studied in other publications [5] and its potential has already been demonstrated.
- Packet aggregation/disaggregation: Ongoing researcher explores the possibilities to accumulate data between a large number of devices (e.g. IoT devices [12]) and aggregate data to send it to the Cloud (and then back). The process followed here encourages the functional split between Edge and Cloud, by offloading some tasks at switches before forwarding traffic data to the Cloud.
- Control plane localization: With P4runtime the control plane for networks at the Edge and networks towards core networks can share the same controller. Indeed, they can also hold local controllers for fast tasks that independently work from centralized controllers.



■ **Figure 4** Local and remote control planes managing different match-action parts of a P4 pipeline.

4 A novel paradigm for Fog Computing

In this part of the paper, we will review how new SDN paradigm controllers can be beneficial for Edge/Fog networks. The controllers can be organized in particular ways to bring some functionalities closer to the Edge and offload others to controllers that are centralized in the Cloud. Edge caching for SDN networks (e.g. answering queries within control plane within P4 switches) is a particularly interesting use case. Other functions like applications that deal with traffic forwarding can work centralized in the network to provide a better path computation calculation. In this section, we address two of the aforementioned areas that are data/control plane partition and Edge caching. In the next paragraphs, we explain how next-generation SDN control planes can be integrated and organized.

In Figure 3 we can observe how we integrate multi-layer SDN control planes. We focus on integrating SDN control planes within the SDN data plane that belongs closest to the network Edge. In this way, we can be sure that locally deployed control planes can manage any functionality that needs faster processing. As shown in Figure 3, we can observe both locally and remotely deployed control planes that are managing P4-based data planes both at the Edge and towards transport networks. With this approach, we enable both locally deployed control plane attend a few functionalities. The same switches also support remotely deployed primary and fallback control planes to attend actions that are delay tolerant but can benefit from a centralized control plane.

While Figure 3 shows how SDN controllers are organized within different network layers, we also want to show how different applications attend functionalities within the same switch. Figure 4 shows how locally and remotely deployed control planes manage their resources (match action units). For instance, as Figure 4 shows, SDN controllers generally act as ARP/NDP proxies, which requires the controller to answer the particular messages. In our example, we decide to move this app to a locally deployed control plane and let it answer messages as requested. Latest OpenFlow-based data planes (e.g. version 1.3 and on) can hold table rules that answer ARP messages too. However, centralized control planes need to attend new ARP requests and prepare the table entries. This now is expected to improve as locally deployed control planes deal with these requests instead of forwarding them to centralized control planes. On the other side, network-wide intents benefit from centralized control planes. To enhance traffic optimization, path computation algorithms benefit from network-wide information instead of locally deployed control planes that exchange routing information.

5 Conclusion

In this paper, we focus on bringing SDN but specifically P4/P4Runtime to Fog networks. While bringing processing to the Edge has many advantages, the challenges that arise need to be considered. To address some of the network challenges, we believe that P4 as a data plane programming language and P4Runtime as a control-to-data plane protocol can benefit next-generation network requirements. Particularly, we propose a new way of deploying SDN control planes that manage both SDN data planes that belong to Fog networks and also SDN data planes that are integrated close to the core network. In this way, SDN controllers can manage cross-layer SDN data planes to offload some functionalities to the Edge (e.g. proxying, caching, small scale auditing, etc.) and also delay tolerant application (e.g. path computation, network wide monitoring etc.). We propose this data plane programming model to serve as the first step into future work to evaluate how next-generation SDN data planes can be distributed and organized to maximize workload for full programmability of next-generation SDN networks. We can demonstrate that this new cross-layer SDN network management is new to Edge/Fog networks and that it can enable a performance improvement of upcoming networks.

References

- 1 Ahmet Cihat Baktir, Atay Ozgovde, and Cem Ersoy. How can edge computing benefit from software-defined networking: A survey, use cases & future directions. *IEEE Communications Surveys & Tutorials*, PP:1–1, June 2017. doi:10.1109/COMST.2017.2717482.
- 2 Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014. doi:10.1145/2656877.2656890.
- 3 P4 Language Consortium. P4. <https://p4.org/>. (Accessed on 02/27/2020).
- 4 The P4.org API Working Group. Specification documents for the P4Runtime control-plane API. <https://github.com/p4lang/p4runtime>, 2020.
- 5 Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. Netcache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th Symposium on Operating Systems Principles, SOSP '17*, page 121–136, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3132747.3132764.
- 6 A. A. Khan, M. Abolhasan, and W. Ni. 5g next generation vanets using sdn and fog computing framework. In *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 1–6, January 2018. doi:10.1109/CCNC.2018.8319192.
- 7 Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008. doi:10.1145/1355734.1355746.
- 8 Jéferson Campos Nobre, Allan M. de Souza, Denis Rosário, Cristiano Both, Leandro A. Villas, Eduardo Cerqueira, Torsten Braun, and Mario Gerla. Vehicular software-defined networking and fog computing: Integration and design principles. *Ad Hoc Networks*, 82:172–181, 2019. doi:10.1016/j.adhoc.2018.07.016.
- 9 F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi. P4 edge node enabling stateful traffic engineering and cyber security. *IEEE/OSA Journal of Optical Communications and Networking*, 11(1):A84–A95, January 2019. doi:10.1364/JOCN.11.000A84.

9:8 Next-Generation SDN and Fog Computing

- 10 J. Pushpa and Pethuru Raj. *Performance Enhancement of Fog Computing Using SDN and NFV Technologies*, pages 107–130. Springer International Publishing, Cham, 2018. doi:10.1007/978-3-319-94890-4_6.
- 11 Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep.*, 1:132, 2009.
- 12 Shie-Yuan Wang, Chia-Ming Wu, Yi-Bing Lin, and Ching-Chun Huang. High-speed data-plane packet aggregation and disaggregation by p4 switches. *Journal of Network and Computer Applications*, 142:98–110, 2019. doi:10.1016/j.jnca.2019.05.008.

Fog Network Task Scheduling for IoT Applications

Chongchong Zhang 

Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences, Shanghai, 200050, China

University of Chinese Academy of Sciences, Beijing, 101408, China

chongchong.zhang@mail.sim.ac.cn

Fei Shen

Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences, Shanghai, 200050, China

fei.shen@mail.sim.ac.cn

Jiong Jin

School of Software and Electrical Engineering, Faculty of Science, Engineering and Technology, Swinburne University of Technology, VIC 3122, Melbourne, Australia

Yang Yang

School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China

Abstract

In the Internet of Things (IoT) networks, the data traffic would be very bursty and unpredictable. It is therefore very difficult to analyze and guarantee the delay performance for delay-sensitive IoT applications in fog networks, such as emergency monitoring, intelligent manufacturing, and autonomous driving. To address this challenging problem, a Bursty Elastic Task Scheduling (BETS) algorithm is developed to best accommodate bursty task arrivals and various requirements in IoT networks, thus optimizing service experience for delay-sensitive applications with only limited communication resources in time-varying and competing environments. To better describe the stability and consistence of Quality of Service (QoS) in realistic scenarios, a new performance metric “Bursty Service Experience Index (BSEI)” is defined and quantified as delay jitter normalized by the average delay. Finally, the numeral results shows that the performance of BETS is fully evaluated, which can achieve 5 – 10 times lower BSEI than traditional task scheduling algorithms, e.g. Proportional Fair (PF) and the Max Carrier-to-Interference ratio (MCI), under bursty traffic conditions. These results demonstrate that BETS can effectively smooth down the bursty characteristics in IoT networks, and provide much predictable and acceptable QoS for delay-sensitive applications.

2012 ACM Subject Classification Networks

Keywords and phrases Task Scheduling, Internet of Things, fog network, delay sensitive

Digital Object Identifier 10.4230/OASICS.Fog-IoT.2020.10

Funding National Natural Science Foundation of China under grant No. 61871370, and Natural Science Foundation of Shanghai under grant 18ZR1437500

1 Introduction

In future Internet of Things (IoT) networks, billions or even trillions of heterogeneous machines and devices are connected by multiple advanced technologies including the Wireless Sensor Networks (WSN), Radio Frequency Identification (RFID), cloud-edge/fog caching and computing [3, 10] and etc. With the acceleration of 5G commercial deployment, a large proportion of the IoT applications is delay-sensitive, such as emergency monitoring, intelligent manufacturing, disaster relief, online games and autonomous driving. The internet traffic of these delay-sensitive applications is bursty and unpredictable at different time scales [8]. For example, some delay-sensitive applications like interactive multiplayer online games, the event-driven applications, intelligent manufacturing and autonomous driving require



© Chongchong Zhang, Fei Shen, Jiong Jin, and Yang Yang;

licensed under Creative Commons License CC-BY

2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020).

Editors: Anton Cervin and Yang Yang; Article No. 10; pp. 10:1–10:9

OpenAccess Series in Informatics



OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

stable delay performance with low delay jitter. In such cases, multipath signals need to be received simultaneously in order to make the next-step control decisions. It is obvious to observe that the QoS of IoT applications depends greatly on the delay performance of the traffic data, which will reflect in the economic benefits of different service providers [11].

However, the wireless communication and computing resources in mobile networks are highly limited, which make difficult to meet the fast growing demands of the booming IoT applications with heterogeneous delay requirements. Therefore, efficient management of network resources and flexible task scheduling algorithms play important roles in both academic researches and industrial applications, especially with bursty task arrivals, dynamic network topologies [2] and unpredictable terminal behaviors [1], in order to guarantee the performance in both the average delay and the delay jitter.

To overcome the severe contradiction between the high delay requirements of the massive traffic generated in the IoT networks and the scarce communication resources, the first thing is to understand the busy characteristics of the data traffic, wherein terminal tasks arrive randomly at the terminal buffers with different arrival rate and task sizes. To take full advantage of the varying characteristic of the wireless channel state in time domain, frequency domain, code domain, etc., plentiful researches are carried out in 5G and IoT networks focusing on the system indexes including the mobility [12, 5], packet delay, and the high frequency transmission [6]. Traditional scheduling algorithms like Proportional Fair (PF) and Max Carrier-to-Interference ratio (MCI) just try to achieve satisfactory bit-level throughput performance [5, 7]. Therefore, novel task scheduling algorithms need to be proposed in order to guarantee the heterogeneous requirements of various delay-sensitive IoT applications with bursty traffic load and dynamic network environments.

The rest of this paper is organized as follows. The system model for terminal task delay are provided in Section 2. Section 3 gives the solution of probability distribution of task delay. The BETS algorithm is proposed in Section 4. Numerical validations are performed in Section 5. Finally, Section 6 concludes this paper.

2 System Model

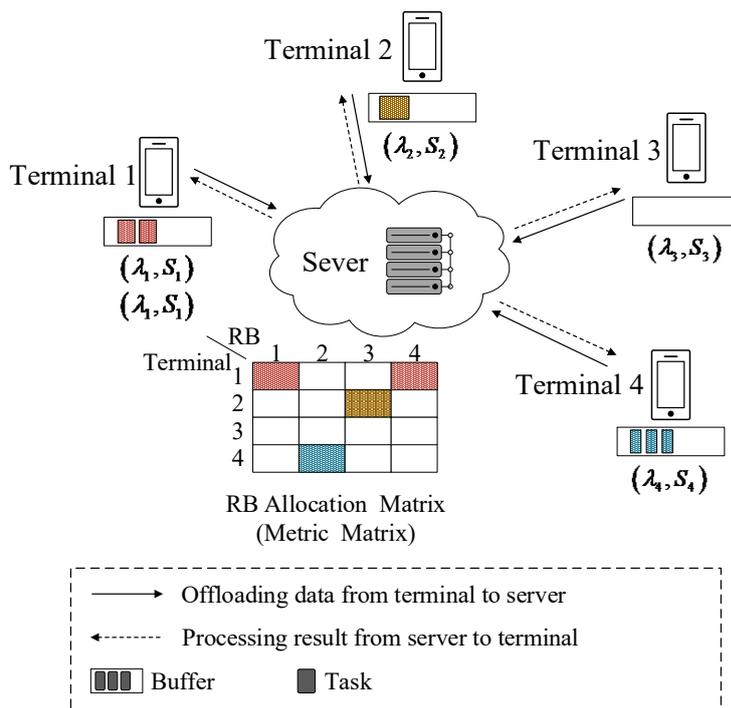
2.1 System Overview

In the traffic layer, an IoT cluster with delay sensitive applications is considered. As shown in Fig. 1, N terminals are randomly distributed in the coverage area of the central server. In the MAC layer, each Time Slot (TS) with duration time of Δt , and the entire system radio resources are divided into M orthogonal parts, i.e. M Resource Blocks (RBs). At the start of each TS, the central scheduler allocates these M RBs to the terminals according to the metric H_{jm} defined by a certain task scheduling algorithm, and the RB will be allocated to the terminal who has the highest scheduling metric on it, which can be formulated as

$$(j^*, m^*) = \arg \max_{j, m} H_{jm}. \quad (1)$$

The task delay in this layer is mainly reflected by the sensing and allocation of the radio resources.

In the physical layer, the instantaneous data rate of terminal j on RB m , i.e. r_{jm} , is set to be Gaussian distributed denoted by $N(E[r_{jm}], \sigma_{jm}^2)$, according to the research on capacity approximation in a Rayleigh fading environment. The data rates of the same terminal on different RBs are assumed to be i.i.d distributed, and the distribution parameters are related to the terminal's position in this cluster [4]. The Probability Distribution Function (PDF) of r_{jm} is $f_{r_{jm}}(x)$. The task delay in this layer is mainly reflected by the transmission delay of the terminal data.



■ **Figure 1** Task scheduling in an IoT cluster with bursty traffic load. The randomly generated terminal tasks are offloaded to the central server through wireless link.

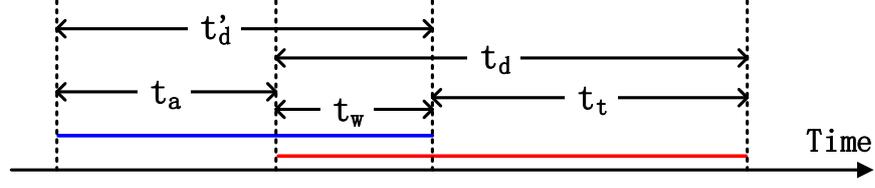
The tasks generated at terminal j follow a Poisson arrival process with arrival rate λ_j and random task size S_j , which arrive at the terminal buffer and need to be offloaded to the central server to be processed. processing delays in the terminal device and the cloud server.

In order to obtain the statistical result of terminal task delay, the offloading processes of the two consecutive tasks of the same terminal and the related parameters are modeled in Fig. 2, where the blue and red lines represent the bustsy timelines of the previous and current tasks respectively. The current task has to wait in the buffer before the delivery of the previous task. The bustsy nature of the traffic and the varying channel state bring challenge to the analysis, which has to take all the traffic layer, MAC layer and physical layer into consideration.

Taking the discreteness of the scheduling process in each TS into consideration and omitting the terminal mark, the time durations in Fig. 2 are defined as follows:

- $t_a = n_a \cdot \Delta t$: the arrival interval, i.e. the time duration between the arrivals of this two tasks;
- $t_w = n_w \cdot \Delta t$: the waiting time, i.e. the time duration between the current task's arrival and its start of transmission;
- $t_t = n_t \cdot \Delta t$: the communication delay, i.e. the time duration between the current task's start of transmission and its delivery;
- $t_d = n_d \cdot \Delta t$: the task delay of the current task, i.e. the time duration between the arrival and the delivery of the task;
- $t'_d = n'_d \cdot \Delta t$: the task delay of the previous task.

The time durations described above are all discrete, and the parameters with the form of " n_* " are nonnegative integers which represent the TS amounts of corresponding time durations. In real systems, the task delay cannot be infinitely large. Thus a threshold $n_{d,\max}$ is defined



■ **Figure 2** The offloading processes of two consecutive tasks of a same terminal.

for n_d , and the probability $\Pr(n_d > n_{d,\max})$ is small enough to be neglected. Therefore, the ranges of these integers are $1 \leq n_d, n'_d \leq n_{d,\max}$, $0 \leq n_w \leq (n_{d,\max} - 1)$, $1 \leq n_t \leq n_{d,\max}$, and $n_a \geq 0$. It is obvious that the probability of the time duration t_* is equivalent to the probability distribution of the corresponding TS amount n_* . Thus we concentrate on the PDF of the delay TS amount n_d when carrying out latter analyses.

3 Problem Solution

To obtain the probability distribution of task delay is equivalent to solving the equation set proposed in the following theorem.

► **Theorem 1.** *The theoretical task offloading delay distribution can be calculated through the following equation set.*

$$V_d^{n_{d,\max} \times 1} = A_t^{n_{d,\max} \times n_{d,\max}} A_a^{n_{d,\max} \times n_{d,\max}} V_d^{n_{d,\max} \times 1} \quad (2a)$$

$$[1 \quad 1 \quad \dots \quad 1]^{n_{d,\max} \times 1} V_d^{n_{d,\max} \times 1} = 1. \quad (2b)$$

In the above expressions, V_d with dimension $n_{d,\max} \times 1$ is the probability distribution vector for the task delay TS amount n_d . A_t and A_a with dimension $n_{d,\max} \times n_{d,\max}$ are the parameter arrays, whose elements are the probability that the task delay is $t_d = n_d \cdot \Delta t$ on condition that the task waiting time is $t_w = n_w \cdot \Delta t$, i.e. $\Pr(n_d | n_w)$, and the probability that the task waiting time is $t_w = n_w \cdot \Delta t$ on condition that the task delay of the previous task is $t'_d = n'_d \cdot \Delta t$, i.e. $\Pr(n_w | n'_d)$, respectively.

Proof. By using the law of total probability twice, the probability for terminal task delay to be n_d TSs, i.e. $\Pr(n_d)$, can be expanded as

$$\begin{aligned} \Pr(n_d) &= \sum_{n_w=0}^{n_{d,\max}-1} \Pr(n_d | n_w) \cdot \Pr(n_w) \\ &= \sum_{n_w=0}^{n_{d,\max}-1} \Pr(n_d | n_w) \cdot \sum_{n'_d=1}^{n_{d,\max}} \Pr(n_w | n'_d) \cdot \Pr(n'_d), \\ &\quad n_d = 1, 2, \dots, n_{d,\max}. \end{aligned} \quad (3)$$

We also have

$$\Pr(n_d) = \Pr(n'_d), \quad n_d = n'_d = 1, 2, \dots, n_{d,\max}, \quad (4)$$

which comes from the fact that for the same terminal, the task delays of all the tasks follow the same statistical probability distribution. For the probability distribution of task delay, i.e. $\Pr(n_d)$, the normalized constraint shown below also needs to be satisfied.

$$\sum_{n_d=1}^{n_{d,\max}} \Pr(n_d) = 1. \quad (5)$$

The vectorial and array representation of the formulas (3) (4) is the formula (2a), and the vectorial representation of the formula (5) is the formula (2b). This completes the proof of the Theorem 1. ◀

4 BETS Algorithm

The traffic load of the terminal in IoT networks is normally a series of tasks, which have dynamic task sizes and randomly generated at the transmitter [9]. The traditional algorithm like PF scheduling tries to pursue satisfactory bit-level throughput, while the QoS of delay sensitive IoT applications draws more concern. All these new features call for newly designed task scheduling policies, which should take into the following considerations.

- Task delay rather than the terminal throughput should become the main scheduling purpose.
- The bursty and heterogeneous characteristics of different applications should be smoothed to provide consistent terminal experience.

Therefore, we introduce the Bursty Elastic Task Scheduling (BETS) algorithm to cope with the bursty nature of IoT traffic.

In a system adopting the BETS algorithm, scheduling decisions are made for all the RBs at the start of TS n . The scheduling metric of terminal j on RB m is defined as

$$H_{jm} = \frac{r_{jm}}{R_j/S_j}, \quad (6)$$

where R_j is the historical average throughput of terminal j . The detailed execution steps of BETS are described in Algorithm 1, and (9) is the updating formula of R_j . The parameter k in the update formula (9) is the average window length, and R'_j is the updated historical average throughput of terminal j . I_{jm} is the indicator variable, the function of I_{jm} is defined as

$$I_{jm} = \begin{cases} 1, & \text{if RB } m \text{ is allocated to terminal } j, \\ 0, & \text{otherwise.} \end{cases} \quad (7)$$

In the scheduling metric of BETS algorithm 1, the terminal with larger task size will have a higher scheduling metric and vice versa, thus a smaller delay jitter can be obtained through BETS. Besides, the BETS is equivalent to the PF scheduling algorithm in the cases that all terminals have the same task size. For comparison, the scheduling metric of PF scheduling algorithm is defined as

$$H_{jm} = \frac{r_{jm}}{R_j}. \quad (10)$$

As for the MCI scheduling algorithm, it directly takes the instantaneous data rate r_{jm} as the scheduling metric and always tries to maximize the system throughput. The execution steps of PF and MCI scheduling algorithms are similar to that of BETS algorithm except for the calculation of the scheduling metric matrix \mathbf{H} .

■ **Algorithm 1** BETS Algorithm.

-
- 1: At the start of each TS, calculate the scheduling metric matrix \mathbf{H} with element H_{jm} in row j and column m as

$$H_{jm} = \frac{r_{jm}}{R_j/S_j};$$

The rows corresponding to the terminals with empty buffers are set to be 0;

- 2: **while** there are nonzero elements in \mathbf{H} , **do**
 3: Find the maximum value in the scheduling metric matrix \mathbf{H} as

$$(j^*, m^*) = \arg \max_{j,m} H_{jm}; \quad (8)$$

- 4: Allocate RB m^* to terminal j^* ;
 5: Set the elements in column m^* to be 0;
 6: **if** Terminal j^* has got enough radio resource to clear its buffer, **then**
 7: Set the elements in row j^* to be 0;
 8: **end if**
 9: Update the historical average throughput of each terminal as

$$R'_j = \left(1 - \frac{1}{k}\right) R_j + \frac{1}{k} \sum_{m=1}^M I_{jm} \times r_{jm}; \quad (9)$$

- 10: **end while**
 11: **return** the scheduling results;
-

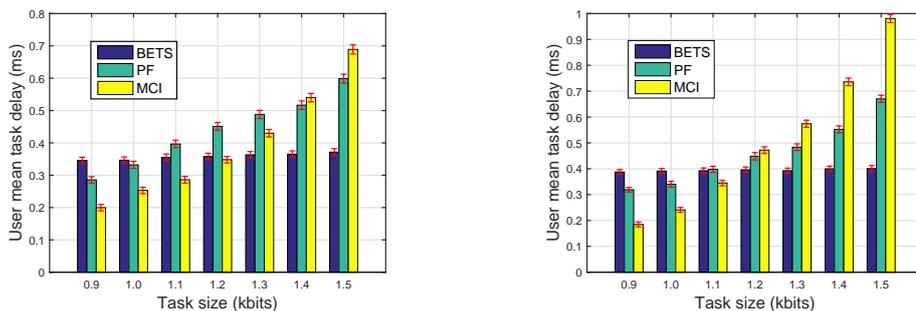
■ **5 Numerical Validations**

In this section, the delay performances of BETS, PF, and MCI scheduling algorithms with varying task size among terminals are investigated in an IoT cluster. In each TS with duration time of 0.1 ms, there are 50 orthogonal RBs to be allocated to multiple terminals according to certain task scheduling algorithms including not only the proposed BETS algorithm, but also the traditional PF and MCI scheduling algorithms. The mean values of the Gaussian distributed instantaneous data rates of the system terminals range from 500 kbps to 1500 kbps as results of the terminals' different positions in the IoT cluster. The average window length of the BETS and PF scheduling algorithms is 500. A total duration of 4 s is set for the simulation process.

In the cases that the task size of the terminals follows Pareto or exponential distributions with mean value of 1 bits, the mean task delays for different task sizes are provided in Fig. 3.

As shown from the numerical results, the BETS achieves a higher fairness for delay performance with varying task size among terminals than those achieved by PF and MCI scheduling algorithms.

For IoT applications with varying task size, it's important to achieve an equalizing delay performance for different task sizes. In the following definition, the the bursty service experience index (BSEI) is introduced to evaluate the delay experience among the system terminals.



(a) Pareto distributed task size with mean value 1 kbits. (b) Exponential distributed task size with mean value 1 kbits.

Figure 3 Terminal mean task delay for varying task size. The theoretical estimation errors are also provided.

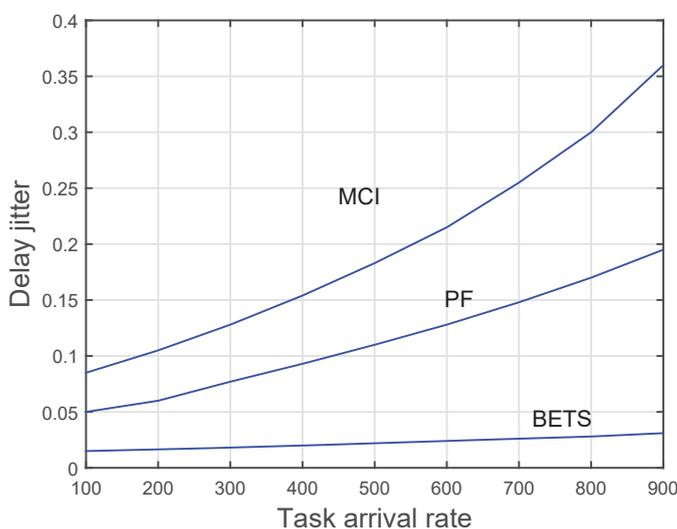


Figure 4 Delay jitter comparisons of the three task scheduling algorithms.

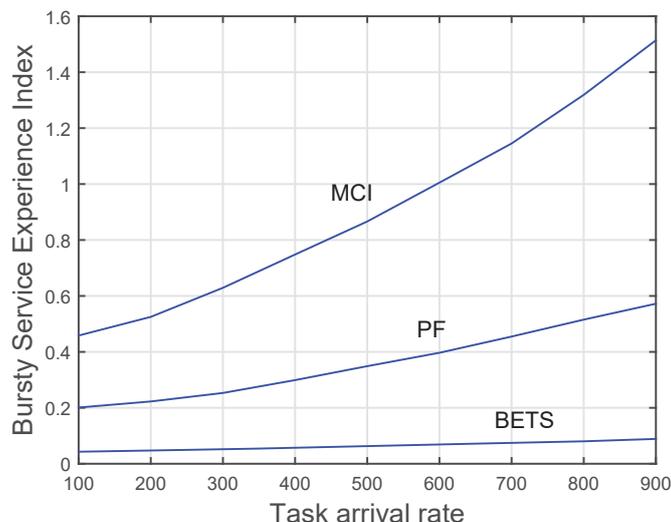
Definition 2. The BSEI of task delay is represented by the following performance index.

$$Q_d = \frac{\sigma(t_d)}{E(t_d)}, \tag{11}$$

where $\sigma(t_d)$ and $E(t_d)$ are the standard deviation and average value of the task delay respectively.

The standard deviation $\sigma(t_d)$ and the BSEI Q_d represent the absolute and relative jitters of the task delay respectively. A smaller Q_d indicates lower jitter and better BSEI for task delay in the system.

The jitter and the BSEI of terminal task delay are shown in Fig. 4 and Fig. 5, respectively, and the BSEI is the ratio of the delay jitter and the average delay as shown in (11). The results in these two figures further validate the superiority of BETS algorithm for achieving better SE and a more consistent performance for terminal task delay, while the other two algorithms achieves much large delay jitter and thus a poor experience for delay-sensitive



■ **Figure 5** BSEI comparisons of the three task scheduling algorithms.

tasks. As shown in Fig. 5, the proposed BETS algorithm can significantly reduce the BSEI, e.g. at a typical task arrival rate of 500, BETS can achieve about 5 times and 10 times more consistent service experience than MCI and PF, respectively, for delay sensitive IoT applications. It's because the introducing of the task size S_j to the scheduling metric of the BETS algorithm. Terminals with higher traffic load have higher scheduling metric as shown in (6).

6 Conclusions

In this paper, the problem of theoretical performance analysis of terminal task delay in IoT networks was investigated. In order to cope with the bursty nature of the traffic statistics in various IoT applications, a novel traffic scheduling algorithm named BETS was introduced, which takes the terminal task size into consideration when making scheduling decisions. Moreover, a new performance metric “Bursty Service Experience Index (BSEI)” is defined and quantified as delay jitter normalized by the average delay to better describe the stability and consistence of Quality of Service (QoS) in realistic scenarios. The numerical results show that the task delay performance of BETS is better than PF and MCI scheduling algorithms.

References

- 1 Najah Abu-Ali, Abd Elhamid M. Taha, Mohamed Salah, and Hossam Hassanein. Uplink scheduling in LTE and LTE-Advanced: Tutorial, survey and evaluation framework. *IEEE Communications Surveys & Tutorials*, 16(3):1239–1265, 3rd quarter 2014.
- 2 Samaresh Bera, Sudip Misra, Sanku Kumar Roy, and Mohammad S. Obaidat. Soft-WSN: Software-defined WSN management system for IoT applications. *IEEE Systems Journal*, PP(99):1–8, 2016.
- 3 Mung Chiang, Sangtae Ha, I Chih-Lin, Fulvio Rizzo, and Tao Zhang. Clarifying fog computing and networking: 10 questions and answers. *IEEE Communications Magazine*, 55(4):18–20, April 2017.
- 4 E. Liu and K. K. Leung. Proportional fair scheduling: Analytical insight under Rayleigh fading environment. In *2008 IEEE Wireless Communications and Networking Conference*, pages 1883–1888, March 2008.

- 5 R. Margolies, A. Sridharan, V. Aggarwal, R. Jana, N. K. Shankaranarayanan, V. A. Vaishampayan, and G. Zussman. Exploiting mobility in proportional fair cellular scheduling: Measurements and algorithms. *IEEE/ACM Transactions on Networking*, 24(1):355–367, February 2016.
- 6 Solmaz Niknam, Ali Arshad Nasir, Hani Mehrpouyan, and Balasubramaniam Natarajan. A multiband OFDMA heterogeneous network for millimeter wave 5G wireless applications. *IEEE Access*, 4(99):5640–5648, 2017.
- 7 Wiroonsak Santipach, Kritsada Mamat, and Chalie Charoenlarnopparut. Outage bound for max-based downlink scheduling with imperfect CSIT and delay constraint. *IEEE Communications Letters*, 20(8):1675–1678, August 2016.
- 8 Philipp Schulz, Maximilian Matthe, Henrik Klessig, Meryem Simsek, Gerhard Fettweis, Junaid Ansari, Shehzad Ali Ashraf, Bjoern Almeroth, Jens Voigt, and Ines Riedel. Latency critical IoT applications in 5G: Perspective on the design of radio interface and network architecture. *IEEE Communications Magazine*, 55(2):70–78, February 2017.
- 9 A. Sivanathan, D. Sherratt, H. H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman. Characterizing and classifying IoT traffic in smart cities and campuses. In *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 559–564, May 2017.
- 10 Yang Yang. Multi-tier computing networks for intelligent IoT. *Nature Electronics*, 2(1):4–5, January 2019.
- 11 Yang Yang, Jing Xu, Guang Shi, and Cheng-Xiang Wang. *5G Wireless Systems*. Springer, 2018.
- 12 Z. Zhang, C. Jiao, and C. Zhong. Impact of mobility on the uplink sum rate of MIMO-OFDMA cellular systems. *IEEE Transactions on Communications*, 65(10):4218–4231, October 2017.

Realizing Video Analytic Service in the Fog-Based Infrastructure-Less Environments

Qiushi Zheng 

School of Software and Electrical Engineering, Swinburne University of Technology,
Melbourne, Australia
qiushizheng@swin.edu.au

Jiong Jin 

School of Software and Electrical Engineering, Swinburne University of Technology,
Melbourne, Australia
jiongjin@swin.edu.au

Tiehua Zhang 

School of Software and Electrical Engineering, Swinburne University of Technology,
Melbourne, Australia
tiehuazhang@swin.edu.au

Longxiang Gao 

School of Information Technology, Deakin University, Melbourne, Australia
longxiang.gao@deakin.edu.au

Yong Xiang 

School of Information Technology, Deakin University, Melbourne, Australia
yong.xiang@deakin.edu.au

Abstract

Deep learning has unleashed the great potential in many fields and now is the most significant facilitator for video analytics owing to its capability to providing more intelligent services in a complex scenario. Meanwhile, the emergence of fog computing has brought unprecedented opportunities to provision intelligence services in infrastructure-less environments like remote national parks and rural farms. However, most of the deep learning algorithms are computationally intensive and impossible to be executed in such environments due to the needed supports from the cloud. In this paper, we develop a video analytic framework, which is tailored particularly for the fog devices to realize video analytic service in a rapid manner. Also, the convolution neural networks are used as the core processing unit in the framework to facilitate the image analysing process.

2012 ACM Subject Classification Computing methodologies → Object detection

Keywords and phrases Fog Computing, Convolution Neural Network, Infrastructure-less Environment

Digital Object Identifier 10.4230/OASICS.Fog-IoT.2020.11

Funding This work was supported in part by the Australian Research Council Linkage Project under Grant LP190100594.

1 Introduction

The rapid development of artificial intelligence, especially deep learning, has shown superior performance in many fields like visual recognition, big data analysis, and natural language processing over the last decade [8]. The convolution neural network (CNN), one of the most preferred deep learning structure, stands out owing to its outstanding performance in data filtering and recognising processes [5]. However, the increasing need for the computational resources for CNN is accompanied by the growing demand for infrastructure support. Specifically, starting at the early LeNet-5 model up to the state-of-the-art InceptionV4 model,



© Qiushi Zheng, Jiong Jin, Tiehua Zhang, Longxiang Gao, and Yong Xiang;
licensed under Creative Commons License CC-BY

2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020).

Editors: Anton Cervin and Yang Yang; Article No. 11; pp. 11:1–11:9

OpenAccess Series in Informatics



OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the accuracy of CNN in image recognition has been improved steadily, yet the computational cost in running the model has also been witnessed a significant increase. Therefore, the use of cloud data-center to execute the CNN model is often considered as the most reliable option in order to tame the complex CNN model.

Alternatively, fog computing, introduced by Cisco and widely used in the edge computing context, is expected to provide the architectural support for various Internet of Things (IoT) services while alleviating the excessive dependence on the cloud [2]. The capability of bringing computation, storage and networking function in the proximity of users thus attracted attention for researchers and industrial practitioners who wish to provision time-sensitive services to the end-users in the IoT environment. Recently, many works have focused on utilizing the fog nodes to implement deep learning for some data-driven services. For instance, Li et al. [6] proposed an approach to offload the first few layers of CNN on the fog nodes for reducing the network traffic from end devices to cloud servers. On the other hand, Ran et al. [12] proposed a framework based on the service requirements to cope with deep learning tasks in either local fog nodes or the remote cloud. Specifically, fog normally plays a cooperative and complementary role with the cloud instead of as a substitute, which means many deep learning approaches still need to be cloud-assisted. However, the cloud is unavailable to cover all deployment scenarios, like the infrastructure-less environment where disrupted or even no electrical grids and cellular networks are the norms.

Taking the national park scenarios in Australia as an example, as the national parks and agrarian business occupy 5.77% and 51% of the Australia land separately [9, 10] while the number is continuing to grow, but only 31% of Australia land has Internet coverage [4] not to mention any form of connection to the electrical grids. The new challenges are then interpreted as the infrastructure-less environments, and it is impractical to continue relying on the cloud due to the unreliable internet connection and unsustainable power supply. Affected by the environmental limit, the implementation of any lightweight application is difficult, not to mention a system that is capable of providing complex video analytic services. Under this condition, infrastructure-less environments urgently need a framework to perform an intelligence video analytic service for protecting the environment, animals and human properties. Therefore, if the service can be realized, a long-term environment preserving strategy could be rolled out by analyzing the relevant information such as the classes of species being recorded or the environmental impact on wildlife.

In short, the contributions of this paper are as follows:

- We identify four key factors that have significant impacts on offloading video analytic services from the cloud to independent fog nodes.
- We propose a fog-based video analytic framework in infrastructure-less environments. The framework achieves a fast running speed by effectively reducing the number of frames to be processed without adversely impacting the accuracy of the results.
- We utilize the Siamese network structure to design a decision approach that is able to process the real-time continuous images based on the similarity efficiently. Consequently, the required computing capability of fog nodes is largely reduced when execution video analytic services.

2 Service Requirements

Recently, some researchers have committed to finding a suitable energy-sustainable fog system that can operate stably for a long period of time while providing valuable information to users in this challenging environment [16]. It is inspiring to provide time-sensitive services

in infrastructure-less environments, but the deep learning-based intelligent services are not provided due to the limitation of the computing capability of fog node. Specifically, video analysing, as one of the widely used fields of deep learning, is considered critical in terms of delivering intelligent services in cases like remote national parks and rural farms in order to achieve the automatic collection and analysis of surrounding information.

Currently, You Only Look Once(Yolo) and Single Shot MultiBox Detector(SSD) are two of the most popular choices to achieve analysing services in most cases. Specifically, Yolo is a framework mostly used for real-time video streaming and demonstrates an excellent performance on boundary detection and object recognition [13]. Meanwhile, SSD plays a significant role in the video analytic area [7]. The essence of these two methods is using bounding boxes to capture many small pieces from the original picture, and then produce feature maps of different sizes through convolution. Finally, each map can be used to predict the targets whose center points are in the small square, which can obtain high recognition accuracy in most working scenarios. These approaches demonstrate a substantial reduction in the calculations compared with window sliding. However, even the light version of Yolo, namely Yolo-tiny, is still considered unworkable on fog nodes without additional hardware support.

Thus, there are four main goals that need to be taken into account so as to realize video analytic services on fog nodes.

Extract Key Frames

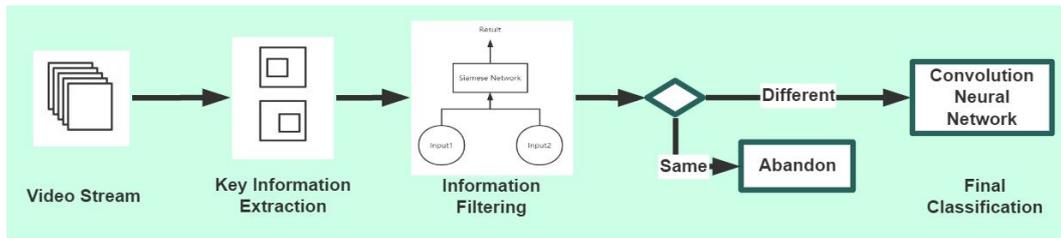
Generally speaking, video data can be treated as a sequence of images with increasing timestamps during real-time processing. The surveillance equipment's output consists of 24-30 frames per second, meaning that a large amount of video data will be transmitted to the computing devices and brings in an unbearable burden to these devices. Hence, the first goal is to reduce the number of output frames to the fog node reasonably and ensure the stability of the fog platform.

Filter Unnecessary Pixels

In order to guarantee the framing range and fidelity of video surveillance, the output is configured to be either 720p or 1080p. However, the input size in deep learning brings undue influence on the fog system, which means that a larger data volume is often accompanied by a much higher requirement in the computation amount. Thus, to ensure the framework having a higher processing rate, the key information on a single image needs to be accurately extracted and appropriately compressed to shrink the input size in the classification neural network.

Acquire Accuracy Results

Using an overly simple neural network structure may enhance multi-frame processing capabilities to a large extent, but it is noticeable that if the video analytic framework cannot provide precise information, gaining higher processing speeds will become meaningless, especially when there is no results correction service provided by the cloud in infrastructure-less environments. For example, early neural network models such as LeNet-5 only consist of a small number of simple convolutional layers. Although the required computational resources required to run the LeNet-5 are moderate, the classification accuracy is unsatisfactory due to the insufficient structure depth. In contrast, state-of-the-art models such as Inceptionv4 are also inappropriate in infrastructure-less environments. These models always concentrated on



■ **Figure 1** The overall architecture of video analytic framework for fog system.

achieving a superior classification without the concerns about the computational resources. In order to ensure the reliability of the classification result, it is thus necessary to adopt the most suitable model that can be afforded by fog devices to extract the key information.

Guarantee Overall Flexibility

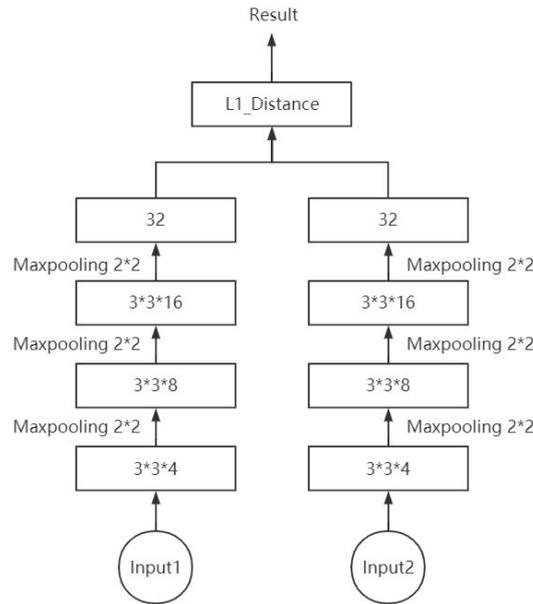
To achieve the video analytic service in the fog-based infrastructure-less environments, a framework is designed to handle this complex situation. The ever-changing residue of computing capability is the most common problem, which is mainly affected by two aspects, one is the number of users allowed to access the fog node services, and the other is the remaining power of the battery. The former will reduce the available computing power in fog nodes, and the impact is still acceptable. The latter will cause severe problems like the power supply shortage of fog nodes or system breakdown. For instance, one common deployment scenario for the fog-based video analytic framework is for wildlife monitoring. As many wild animals are nocturnal animals, the framework thus needs to perform well at night and expects to execute at the lowest working state of the CPU to reduce the consumption of stored electricity. Therefore, the video analytic framework should adapt to different computing capabilities to retain satisfactory services in different working scenarios.

3 Video Analytic Framework

In order to address the aforementioned issues and provide video analytic services in the infrastructure-less environment, we proposed a framework to splits a video processing process into multiple parts and optimizes them separately. As shown in Fig. 1, the framework consists of three main parts, including key information extraction, information filtering, and final classification.

3.1 Key Information Extraction

Due to the lack of computation power in fog nodes, the first priority is extracting the key images from the video stream so that the total amount of images that needed to be processed remains low. In infrastructure-less environments, the animals might appear in the captured image and stay for a short period. Thus, the frame difference method is considered as the most suitable approach to obtain several key frames from a series of video frames in which the animals appear. Specifically, the frame difference method is to subtract the pixel values of two images in two adjacent frames or a few frames apart in the video stream to extract the moving area in the image [15]. The benefits of this approach mainly from two aspects: Firstly, because of the sensitivity to moving objects, the appearance and movement of animals can be accurately captured while the frame in the stationary state can be ignored automatically.



■ **Figure 2** A customized 4-layer Siamese neural network structure.

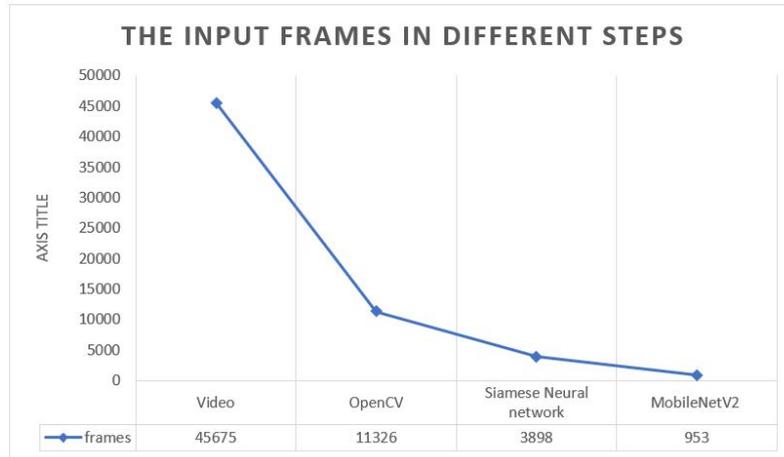
Compared with other background extraction algorithms, the frame difference method has a better balance between the processing performance and computing consumption in the animal detection scenario. Secondly, it has high tunability, the time between two frames can be adjusted dynamically based on the computation resource of fog nodes to acquire optimize processing speed and avoid excessive battery drain.

However, in actual tests, we found that the information obtained by the frame difference method is not always accurate, which is often caused by the movement of only a part of the animal's body, like feet or tails. In order to solve this problem, we empirically enlarged the size of the acquisition area when generating small-scale image changes to ensure the integrity of the image as much as possible. After that, the video data is then able to be converted to a series of clear animal pictures, and all pictures can be quickly scaled down to the same resolution (224*224) for further processing.

3.2 Information Filtering

In most cases, extracting key information from the video stream can solve the problem of insufficient computing power due to the significant reduction of images that need to be processed, but it is far from enough in infrastructure-less environments. Through the analysis of the obtained pictures, we found that the pictures have a high degree of similarity because it contains the same animals with various behaviors in a short time. If the pictures containing the same animal with different actions can be distinguished clearly, it can further decrease the number of pending pictures waiting to be processed and maximally save computing resources. Therefore, we introduced a neural network structure serving for picture filtering, called Siamese neural network.

The idea of the Siamese neural network is to learn a function that differentiates two similar inputs through two neural networks with shared parameters [3]. Different from the traditional neural network in image classification, the Siamese neural network only infers whether two input objects belong to the same type, and the output is "same" or "different" instead of the class.

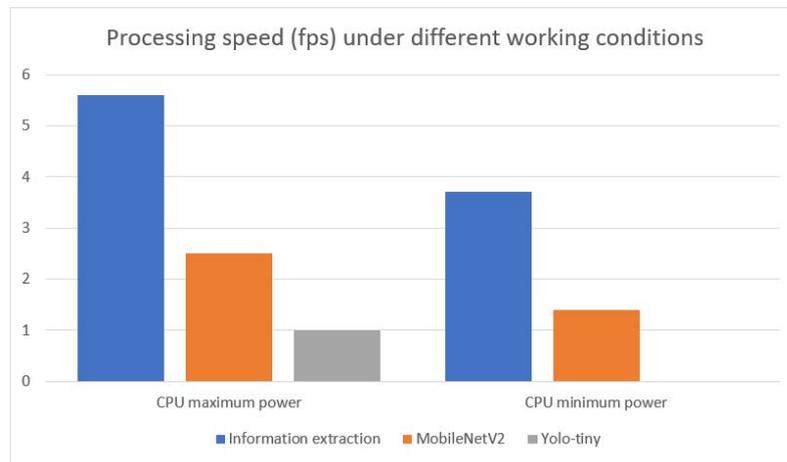


■ **Figure 3** The number of critical frames obtained after each extraction step.

As shown in Fig. 2, we implemented a 4-layer Siamese neural network structure and use L1 distance to measure the similarity. Since the image resolution obtained from the first step is $224 * 224$, we need to compress it to $56 * 56$ before sending it to the Siamese neural network. At the same time, we adjusted the structure of the traditional Siamese neural network. From the system's perspective, the pictures received from the first step are time-continuous, and we do not need to submit two frames at the same time for similarity comparison. If it is determined that the two inputs belong to the same animal, the Siamese neural network will abandon the first frame and retain the operation result of the second frame, then output the classification result of the first frame. Conversely, if the two frames are different animals, the network will upload the second frame to the next step to obtain the classification result. In order to verify the performance, we generated 30,000 pairs of training data and 6,000 pairs of verification data using pictures of 5 different birds obtained from the bird observation video to complete the module training that the accuracy could reach up to 99.4%. The trained model has achieved the highest possible accuracy on the specified training and testing data sets, but this result is limited to the five bird species used in the data sets due to the similarity of these two data sets. Therefore, the images of other animal species that have not been contained in the data sets are suitable to verify the robustness of the Siamese network model. To this end, 1757 consecutive pictures of pet dog activity are gained by the key information extraction from a 4-minute video, and then are submitted to the model for acquiring decision results. The Siamese network model successfully achieves more than 75% accuracy on untrained animal classes, which shows an exceptional potential in information filtering.

3.3 Final Classification

Through these two processing tasks, the video stream data is converted to a small group of distinguished pictures, and the final task is to choose an appropriate neural network model for the classification based on the computing capability of fog nodes. In order to ensure the reliability of the classification results and the deployability on fog nodes, we choose MobileNetV2 as the model to complete the image classification task. Specifically, MobileNetV2 is tailored for mobile and resource-constrained environments, which retain the same accuracy with a significant decrease of numbers of operations and memory needed [14].



■ **Figure 4** The processing frame rate in different working conditions.

We train the model with an open dataset from Kaggle, which contains 25,000 dog and cat images. For reducing the training time and testing the generalization ability of the model, we firstly use the transfer learning technique to freeze all convolution layers based on the pre-trained Imagenet model and only train the full connection layers. MobileNetV2 unsurprisingly achieved a high accuracy rate that around 96.07%.

4 Experimental Results

In this section, we completed a series of experiments on Raspberry Pi 3B+ (RPI) to demonstrate the framework performance in infrastructure-less environments.

OpenCV, as a lightweight and efficient cross-platform computer vision library [11], has been installed in RPI to implement the frame difference method, and the interval between two frames is controlled to adjust the occupied computing resource. Besides, the Siamese neural network and MobileNetV2 are established by Tensorflow, which is an open-source software library to fulfill different machine learning tasks [1]. Afterward, we downloaded animal videos from YouTube, which were collected by fixed-position cameras. In order to make the experimental results in line with the actual environment, we pre-processed the resolution and frame number of the video to obtain the same parameter settings as the surveillance camera, 720p and 25fps.

Fig. 3 demonstrates the number of critical frames obtained after each extraction step. It can be observed that the total number of frames has decreased significantly, and only a few pictures need to be processed on MobilNetV2. Furthermore, the initial frame resolution in the video stream is 1280*720, and the output images from OpenCV are compressed to be 224*224 that only contains critical information. For evaluating the performance, we used 6 videos with a total duration of 1,827 seconds and the total frame number of the video is 45,675 frames. Since 25 frames per second have greatly exceeded the processing capacity of OpenCV on RPI, the frame difference method will jump 2 frames in order to guarantee the stability of the framework. In other words, it will execute every 4 frames at each timestamp. In the experimental video, OpenCV has executed 11,326 times and identified 3,898 frames containing moving animals. After the similarity judgment, only 953 frames that occupy 2.08% in the total video frames need to be passed from the Siamese neural network to the MobileNetV2 for acquiring all classification results in the video.

At the same time, to provide long-term operation in infrastructure-less environments, the performance of the framework under different power consumption states should be evaluated. Fig. 4 shows the maximum processing speed can be achieved when the framework realizes real-time services under different working states of the CPU. Obviously, the performance of the framework is satisfactory even operates with minimal CPU power, and the processing frame rate is much better than deploying Yolo-tiny.

5 Conclusion

In this paper, we propose a fog-based framework in infrastructure-less environments to achieve the video analytic service. By utilizing the frame difference method and the Siamese neural network to extract the key information in the video, the video analytic framework successfully converts the huge amount of video data that fog nodes cannot afford into a small amount of critical image. Specifically, the framework overcomes the computation limitation in fog nodes to obtain classification results and minimizes the usage of computing-intensive CNN. Additionally, the experimental results clearly show a good performance of our proposed video analytic framework and its capability to deal with emergencies by distributing tasks to other fog nodes due to the shrink of input data size. Our next phase of research will focus on developing the communication strategy to decide the assignment of tasks among nodes in real-time to obtain better processing capabilities.

References

- 1 Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, November 2016.
- 2 Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16, August 2012.
- 3 Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546. IEEE, June 2005.
- 4 Australian Competition & Consumer Commission. Domestic mobile roaming declaration inquiry final report, 2017.
- 5 Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, pages 1097–1105, 2012.
- 6 He Li, Kaoru Ota, and Mianxiong Dong. Learning iot in edge: Deep learning for the internet of things with edge computing. *IEEE Network*, 32(1):96–101, January 2018.
- 7 Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, pages 21–37. Springer, 2016.
- 8 Lingjuan Lyu, James C Bezdek, Xuanli He, and Jiong Jin. Fog-embedded deep learning for the internet of things. *IEEE Transactions on Industrial Informatics*, 15(7):4206–4215, July 2019.
- 9 Australian Government Department of Agriculture. Land use in australia at a glance 2006. URL: https://www.agriculture.gov.au/sites/default/files/abares/aclump/documents/Land_use_in_Australia_at_a_glance_2006.pdf.
- 10 Australia Government Department of the Environment and Energy. Capad 2016 national summary. URL: <https://www.environment.gov.au/land/nrs/science/capad/2016>.

- 11 Kari Pulli, Anatoly Baksheev, Kirill Korniyakov, and Victor Eruhimov. Real-time computer vision with opencv. *Communications of the ACM*, 55(6):61–69, June 2012.
- 12 Xukan Ran, Haolanz Chen, Xiaodan Zhu, Zhenming Liu, and Jiasi Chen. Deepdecision: A mobile deep learning framework for edge video analytics. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 1421–1429. IEEE, April 2018.
- 13 Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, June 2016.
- 14 Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, June 2018.
- 15 Nishu Singla. Motion detection based on frame difference method. *International Journal of Information & Computation Technology*, 4(15):1559–1565, 2014.
- 16 Qiushi Zheng, Jiong Jin, Tiehua Zhang, Jianhua Li, Longxiang Gao, and Yong Xiang. Energy-sustainable fog system for mobile web services in infrastructure-less environments. *IEEE Access*, 7:161318–161328, 2019.

Detection of Fog Network Data Telemetry Using Data Plane Programming

Zifan Zhou 

DTU Fotonik, Technical University of Denmark, Lyngby, Denmark
zifz@fotonik.dtu.dk

Eder Ollora Zaballa 

DTU Fotonik, Technical University of Denmark, Lyngby, Denmark
eoza@fotonik.dtu.dk

Michael Stübert Berger 

DTU Fotonik, Technical University of Denmark, Lyngby, Denmark
msbe@fotonik.dtu.dk

Ying Yan 

DTU Fotonik, Technical University of Denmark, Lyngby, Denmark
yiya@fotonik.dtu.dk

Abstract

Fog computing has been introduced to deliver Cloud-based services to the Internet of Things (IoT) devices. It locates geographically closer to IoT devices than Cloud networks and aims at offering latency-critical computation and storage to end-user applications. To leverage Fog computing for computational offloading from end-users, it is important to optimize resources in the Fog nodes dynamically. Provisioning requires knowledge of the current network state, thus, monitoring mechanisms play a significant role to conduct resource management in the network. To keep track of the state of devices, we use P4, a data-plane programming language, to describe data-plane abstraction of Fog network devices and collect telemetry without the intervention of the control plane or adding a big amount of overhead. In this paper, we propose a software-defined architecture with a programmable data plane for data telemetry detection that can be integrated into Fog network resource management. After the implementation of detecting data telemetry based on In-Band Network Telemetry (INT) within a Mininet simulation, we show the available features and preliminary Fog resource management based on the collected data telemetry and future telemetry-based traffic engineering possibilities.

2012 ACM Subject Classification Networks → Programmable networks

Keywords and phrases SDN, P4, P4Runtime, control planes, Fog, Edge

Digital Object Identifier 10.4230/OASICS.Fog-IoT.2020.12

1 Introduction

Transmission latency for IoT applications such as health monitoring and emergency response has become crucial in providing reliable and efficient performance. However, due to the distributed location of IoT devices, some may be far from the core and Cloud networks. Considering that the latency of data communication is significantly impacted by the distance, IoT data can experience long propagation delays to reach the Cloud [3]. Besides, the centralized Cloud data centers often store and process a large amount of data from billions of IoT devices, the heavy workload can also cause a long processing delay for IoT data. Fog computing enables the distribution of Cloud services to the edge network, with a closer location to the devices. Therefore, the round trip latency from devices to the Fog and back to the devices is shorter. Fog networks are usually lightweight, and several Fog nodes could be placed at the edge network separately, the possibility of network congestion is lower, by



© Zifan Zhou, Eder Ollora Zaballa, Michael Stübert Berger, and Ying Yan;
licensed under Creative Commons License CC-BY

2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020).

Editors: Anton Cervin and Yang Yang; Article No. 12; pp. 12:1–12:11

OpenAccess Series in Informatics



OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

distributing data to these Fog nodes. Last but not least, regarding the energy needed to transmit a byte of data from IoT devices to computing nodes, processing applications expect that Fog Computing can also reduce the energy consumption in the network [11].

To maintain a high availability and performance of Fog computing, resource provisioning plays an essential part in network management. These decisions are usually made based on specific metrics to distribute workload optimally. Factors such as time, energy, user-application context, etc., have been mentioned in the literature of Fog computing. Since latency-critical applications have specified a tolerant maximum delay for finishing time, it is straightforward to provide service and resources according to network delay, i.e. the whole time for completing a task should be ahead of the deadline of the application. Unlike Cloud computing where most of the nodes are closely located in the data center, Fog nodes are often dispersed over the access network, which adds an extra transmission delay among Fog nodes, and that makes it more difficult for Fog nodes to make offloading decisions. Accordingly, to fulfill the transmission of Fog data, workload distribution has to take the offloading delay into consideration, which can be collected by the application of data plane programming when all nodes are time synchronized.

Integrating a centralized controller into a distributed Fog architecture further facilitates network management. Given the data telemetry of the Fog network, the controller sends real-time workload distributing instructions to linked nodes. The implementation of a centralized control plane assesses all delay factors and with the ultimate goal to fulfill transmission requirements, avoiding a lack of status information due to an isolated location in Fog networks.

In the existing literature, the research community has proposed several studies on the delay analysis of Fog computing, but there are not many discussions on detecting and monitoring the Fog network state in practice. In [10], mathematical formulas of computation and transmission delay in Fog computing and one offloading mechanism to optimize the resource allocation were introduced. Another mathematical model of latency in Fog architecture for 5G cellular networks was proposed in [4]. In [13], a software-defined embedded system for Fog computing optimization was presented. In [6], the authors proposed a delay-aware path finding algorithm based on a Software-Defined Network (SDN) framework and OpenFlow protocol to optimize network routing/rerouting performance.

Following the motivations and the existing research, we propose an architecture using programmable data planes to collect data telemetry and conduct workload balance, moreover, with a SDN controller to perform centralized management of the Fog network. We use queuing delay as the metric for resource management in the architecture. Finally, we create a simulation of the network monitoring and load balancing in Mininet [8] for a proof of concept. The rest of the paper is organized as follows: Section 2 explains the data plane monitoring and the proposed architecture. Section 3 includes the implementation of the simulation in Mininet and the topology used for the simulation. In Section 4, we present the results of the collected data telemetry, and finally, in Section 5 we conclude the paper and discuss the future work.

2 Software-defined managing architecture

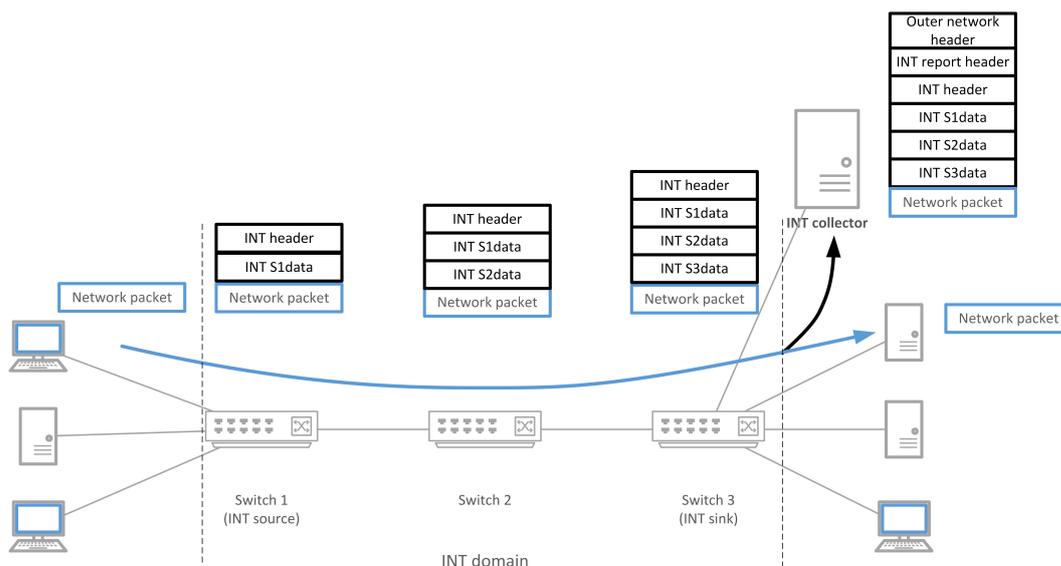
In this section, we briefly describe the architecture for data telemetry detection and Fog resource management. To keep track of the network state, a few protocols have been used to request and present the real-time state of the network devices, e.g. Simple Network Management Protocol (SNMP) [12]. In this proposal, we leverage the data plane programming

language P4 [1] for the devices used in Fog networks (i.e. gateways, switches, and Fog computing devices, etc.). P4 programs are used to define the pipeline behavior for packet processing in all the P4 programmable devices and the control plane can manage the data plane via a control-to-data runtime protocol (P4runtime).

By using P4 programs in Fog networks, specific network telemetry can be collected via data-plane operations. This is achieved via P4 programming and In-band Network Telemetry (INT) [7]. The devices create metadata of each packet within the pipeline, therefore, it is possible to collect internal timestamps when entering and leaving the queues of the pipeline. Meanwhile, the queue depth metadata changes along receiving and sending packets, which indicates the current workload of the device.

INT was derived from the Tiny Packet Program (TPP) that embeds programs into network packets from end hosts to query the state of network switches [5]. Similarly, INT headers that contain INT instructions and description of the header are added to normal network packets.

As shown in Figure 1, we define the INT domain which contains all the P4 programmed INT-capable devices. The process is initiated by the first node in the domain, so-called *INT source*, which adds the INT header and the required data telemetry to a particular packet passing being forwarded. The header can be encapsulated as a payload after several network protocols (e.g. after TCP or UDP and before payload, etc). The following devices on the path interpret the instruction contained in the INT header and then add an extra layer of telemetry data per device with the corresponding state information to the telemetry data within the packet. After the last device in the INT domain (i.e. the *INT sink*), the generates an *INT report* by encapsulating the packet that has just traversed the network. When the packet has been cloned in the switch pipeline, the *INT sink* also removes the telemetry headers and data to recover the original network packet from and sends it to the receiver. Therefore the *INT sink* generates 2 packets, one for the original traffic endpoint (without any telemetry data attached) and the other as an *INT report* packet for the telemetry collector.



■ **Figure 1** Illustrating of data telemetry collection in INT-capable P4 devices.

As mentioned in the previous paragraph, the destination address in the outer network headers of the *INT report* refers to the collector where all the INT packets are sent to. The collector is an endpoint that decodes all the INT header stacks and, in our proposal, stores delay information and tracks the real-time changes. Furthermore, the collector can make decisions to distribute the workload among Fog nodes based on the delay information. Thus, it requires several modules and interaction among different tasks to assemble the collector, and in this architecture, we use a Software-Defined Network (SDN) controller that integrates all the modules and comes with the following benefits:

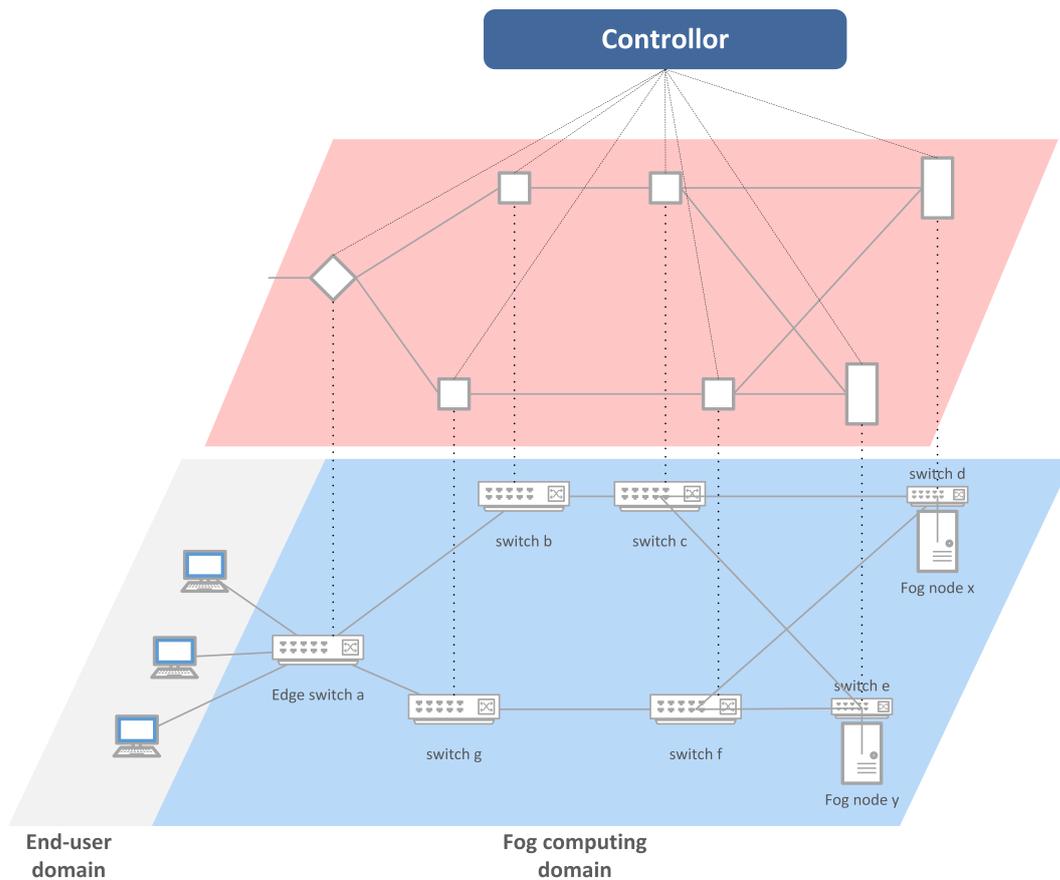
- **Centralized Fog management:** the control plane is separated from the data plane in Fog networks, the abstraction and control of the network are managed by the controller.
- **Real-time network management:** implementing the SDN controller and a programmable data plane in Fog networks enables controllers to modify flow tables in the devices at runtime.
- **Flexible application:** controller's network abstraction and shared control-plane software modules make it easier to program network applications such as routing algorithms and load balancers in the Fog network.

Figure 2 depicts the abstraction of an example Fog network with the architecture. As mentioned above, we used a P4 program for the data plane of the devices used in the Fog network. All of the devices are physically connected with the controller, thus the controller has direct access to modify the data plane actions of the devices. Historically, communication between the controller and the devices is achieved by a well-defined application programming interface (API), OpenFlow [9]. However, in our example, we use P4Runtime as the protocol between the centralized controller and switches. Using P4Runtime allows controller configurations to stay both centralized for various switches or work locally within a single switch (which can not be done using OpenFlow-enabled switches). Having control planes that can work independently within the same switch and pipeline can help in developing future applications for data plane telemetry and traffic steering. For instance, a local control plane can be in charge of small scale monitoring while a centralized control plane can be in charge of computing new paths using a centralized network view. The tasks can be properly organized among different controller configurations to achieve better performance.

By sending INT instructions from the controller, the *edge gateway a* can initiate the INT process, and the last-hop switch, e.g. top-of-rack (TOR) switch, to the Fog node, will generate the *INT report* or vice versa. To follow our future goals of using telemetry data, the queuing in both directions, as well as processing delay (within the pipeline) in the Fog nodes, are exposed to the controller, then it can improve traffic forwarding paths following a path optimization using telemetry data.

To collect the network state, monitoring protocols either create special probe packets (postcard mode) or add extra headers (integrated into traffic), both methods increase the overhead of the network. On the other hand, the precision of the detection is dependant on the frequency of collecting of network state, but frequent detection usually generates more overhead. In this architecture, we set the INT frequency adjustable considering the trade-off with overhead, thus, not all the network packets carry the INT header and the results can describe the network state. Only packets that match specific rules on a table at the beginning of the pipeline will ever be monitored, leaving the rest of the traffic not tracked.

In this architecture, data telemetry in the network is detected by the centralized accumulator, hence, the SDN controller could also distribute computational resources according to the real-time network state. Since the INT packets only traverse within the INT domain and through data plane, end devices and applications in the Fog nodes are unaware of the whole detecting process.



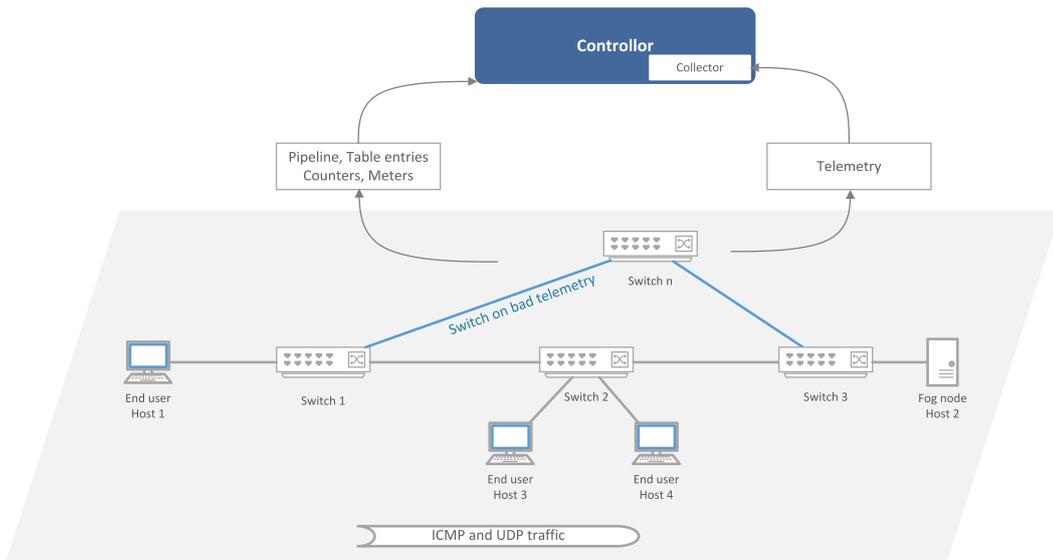
■ **Figure 2** All devices in the Fog computing domain are connected to the controller, and all the INT reports are sent to the controller, thus the controller has the abstraction of the network and the state of each link and device.

3 Simulation

In this section of the paper, we have tried to demonstrate, within the Mininet simulation environment, the telemetry capabilities of INT and P4 programmable devices. In Mininet we set up virtual hosts as nodes that generate and receive packets. P4-compatible BMv2 [2] software switches are used as programmable switches in the simulations.

In these tests, we have set up 4 switches in between one Fog node (Host 2) and an end-user (Host 1) to test the time that packets spend in switches' queues. The first test shows attempts with continuous ICMP traffic exchange so we can observe a specific period of the time that packets spend in queues. In the second test, we conduct the same simulation but test with UDP traffic exchange using iperf. We try to load the switches with a relatively higher demand for traffic (1 Mbit/s of bandwidth, 1000 bytes per packet) and present the performance of the switches from queuing delay telemetry. We have to lower the maximum packet size in the generated UDP stream in order to be able to attach telemetry data to the network packets (without surpassing maximum Ethernet packet size).

In the last test, we demonstrate the preliminary redirecting of network flow in the simulation when one switch on the path starts to show poor performance. During the whole simulation period, Host 1 keeps sending ICMP traffic to Host 2 through the default path:



■ **Figure 3** Simulation topology to monitor traffic between edge switch and end user. This telemetry data can be reused to update the traffic forwarding at runtime.

switch 1 – switch 2 – switch 3. Then we create the scenario when switch 2 has a descent of performance by sending interfering traffic from Host 3 to Host 4 through switch 2. The queuing delay is monitored along with the simulation and after reaching a certain threshold, i.e. bad telemetry, switch 1 will redirect the ICMP flow to switch n, hence offloading from switch 2.

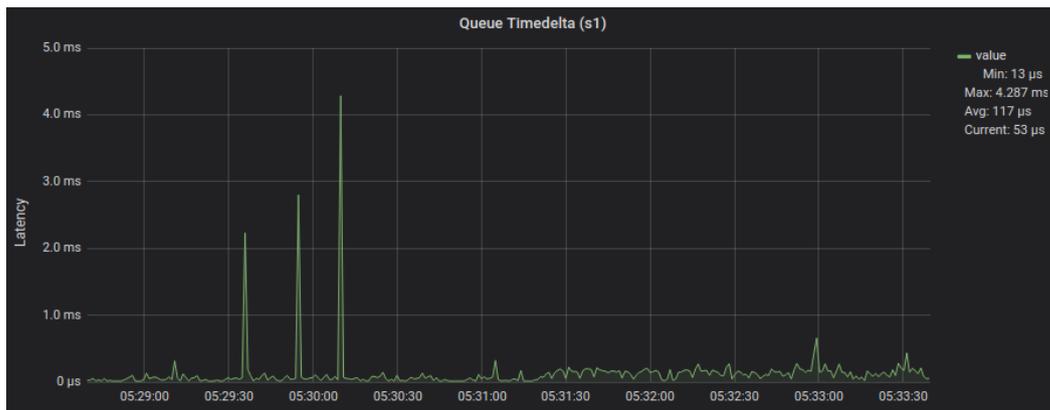
As seen in Figure 3, we have used the Mininet emulator to create a virtual network that connects virtual hosts (Edge/Fog devices) with virtual P4 programmable switches. This enables us to create custom topologies and test traffic between hosts with a custom network pipeline. In this way, we have been able to customize the pipeline and track which network flows we want to monitor. In this case, we have both tracked the ICMP requests from Host 1 (IP=10.0.1.1) to Host 2 (IP=10.0.2.2) and also the specific *iperf* generated UDP traffic from Host 1 (*iperf* client) towards port 4444 on the server. In order to monitor these flows, we program the tables of the pipeline by defining which parameters of the flows we monitor. Specifically, fields of IPv4, ICMP and UDP headers.

When we detect a packet from a type of traffic that we have to monitor, we add the telemetry headers and extract them at the end of the path, sending the whole packet and telemetry headers to a collector. The data stored by the collector is the preferred data needed by the controller to, possibly, change traffic flows at runtime.

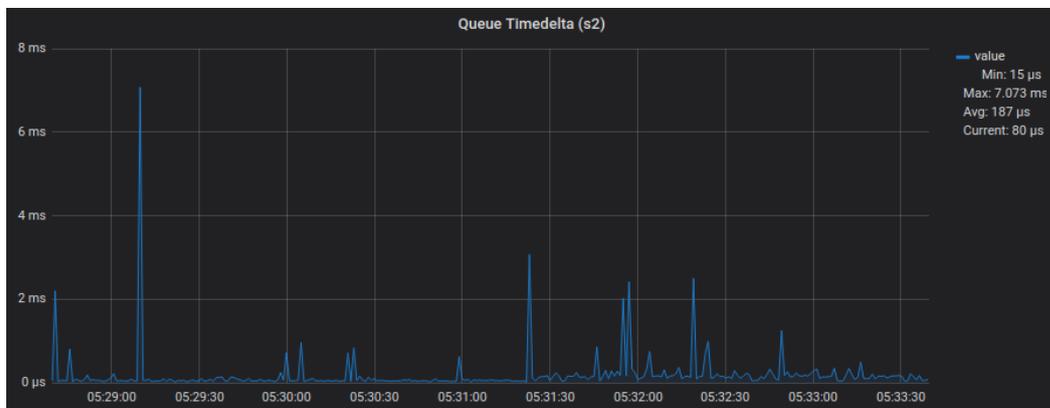
4 Results

In this part of the paper, we focus on bringing simulation and results for INT monitoring. As we have set up the environment to monitor the state of P4 programmable switches, now we present the statistics of telemetry to demonstrate how we can visualize the internal state of forwarding devices to possibly further use it with traffic engineering purposes.

In Figures 4 and 5, giving the results when the ICMP request packets circulating over S1 and S2 from Host 1 to Host 2, different behavior of both switches are shown. In Figure 4 we can observe 3 outlining spikes that define a long time of packets within the switch but overall



■ **Figure 4** Time that packets spent in the queue on Switch 1 for ICMP traffic exchange. Represents ICMP requests from Host 1 to Host 2.



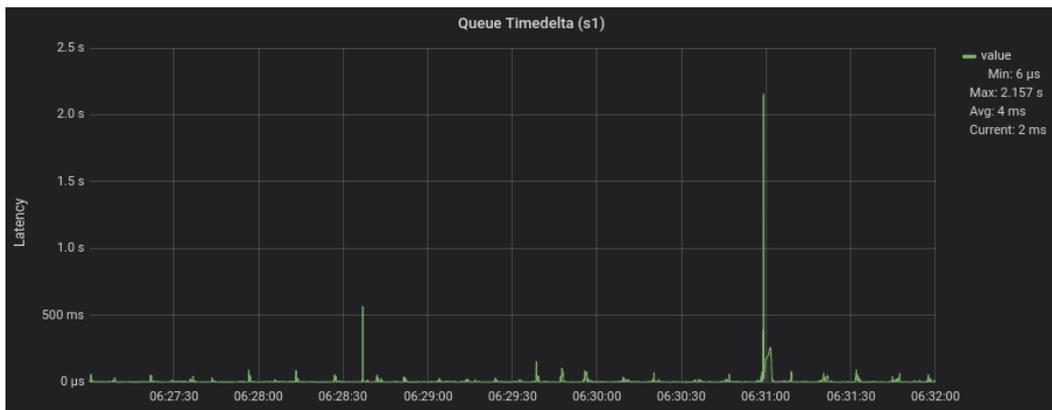
■ **Figure 5** Time that packets spent in the queue on Switch 2 for ICMP traffic exchange. Represents ICMP requests from Host 1 to Host 2.

good behavior considering the rest of the data. The switch shows 3 spiking queue delays over 2 ms but an overall correct queue delay. We have a similar result in Figure 5, but this time, there is an outlining spike at the beginning of the figure and then a few less well-performing spikes exist in the rest of the figure. ICMP packets are not very demanding for switches so packets spend a very short time in queues, graphs show an average time of 117 and 187 μ s, respectively. Seeing these average results and Figures 4 and 5 we can observe that S1 has performed slightly better than S2. As we have not stress-tested the switches, the spikes on S1 and S2 only refer to the unexpected bad performance that software switches could have.

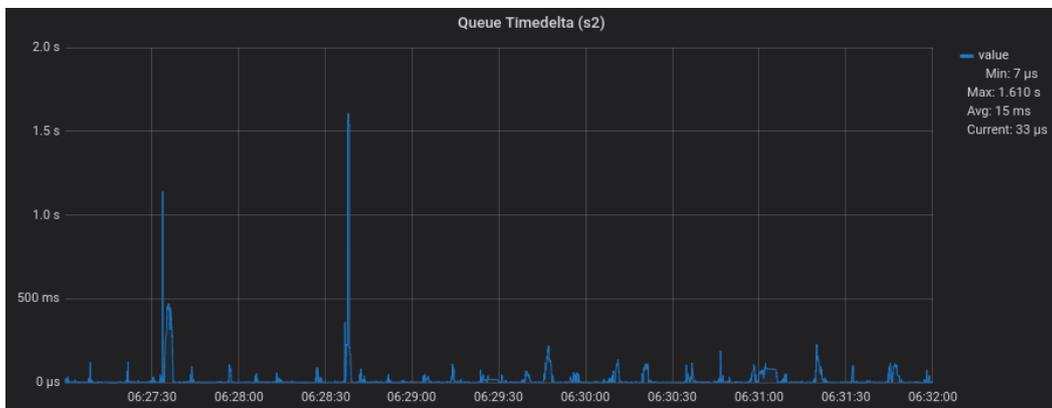
Looking into Figures 6 and 7, we can demonstrate the ability to monitor and signalize badly performing moments of the switches. For instance, Figure 6 shows 2 specific moments when packets experience relatively long queuing time in the switch. The peek in the result gives a worst-case queuing time for over 2 seconds, which can result in a poor experience for a realistic latency-critical application that might be harmful. Due to the amount of traffic, switches need to perform tasks quicker and faster than in the previous case, having specific bad performing moments.

Similar problems appear in Figure 7, where the worst-case results show a few cases of 1.5 seconds queuing time. Because the BMv2 switch is a virtual switch, it shares some capabilities with the virtual machine (VM) that hosts the testing process. We can expect a

12:8 Detection of Fog Network Data Telemetry Using Data Plane Programming



■ **Figure 6** Time that packets spent in the queue on Switch 1 for UDP traffic exchange. Represents packets from Host 1 (iperf client) to Host 2 (iperf server).

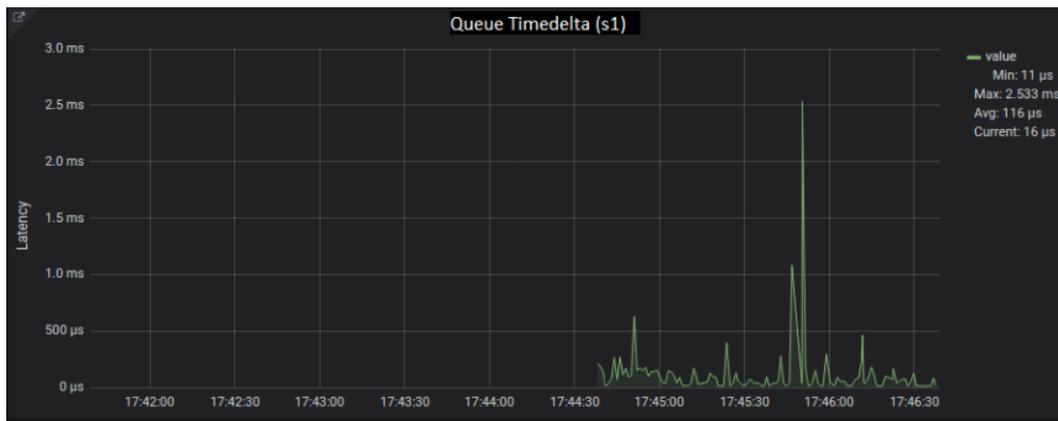


■ **Figure 7** Time that packets spent in the queue on Switch 2 for UDP traffic exchange. Represents packets from Host 1 (iperf client) to Host 2 (iperf server).

few cases that the host VM might affect the switch performance in the simulation process, like the highest peaks shown in Figures 6 and 7, whereas learning from most other packets, switches have performed correctly. Besides having a higher amount of traffic compared to Figures 4 and 5 shows an expected worse performance in Figures 6 and 7.

Figure 8 illustrates the queuing delay from switch 1 in the flow redirecting scenario. As we can see, ICMP flow sent by Host 1 gets received in the queue of switch 1 at around time 17:44:40. The spikes in the queuing delay of switch 1 are mainly caused by the instability of the software switch used in the simulation. The ICMP flow shows up at switch 2 right after the moment it leaves switch 1, as given in Figure 9. From time 17:45:20, Host 4 starts the transmission of interfering flow to switch 2, and the queuing delay shows three significant increases due to each time the burst of interference. After the last spike at time 17:46:10, the redirecting threshold is reached and the ICMP flow no longer appears in the queue of switch 2. Instead, as shown in Figure 10, since time 17:46:10, the flow is redirected to switch n, indicating the workload has been distributed from switch 2 at runtime.

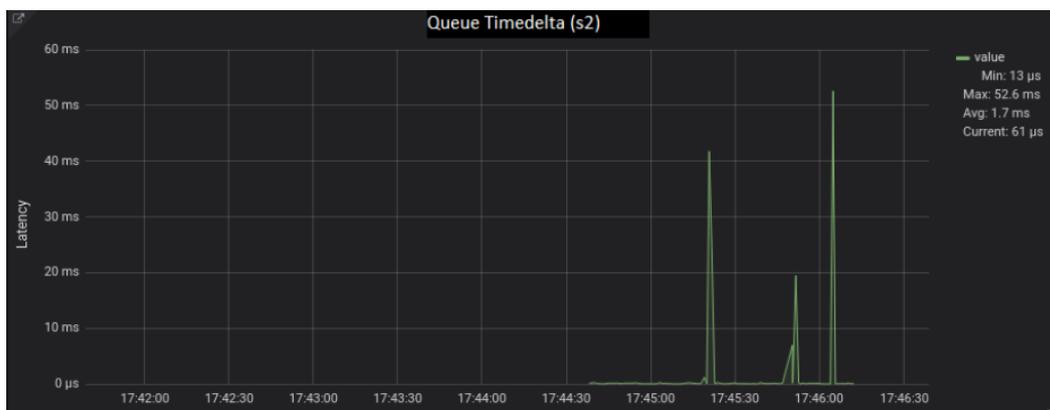
These tests demonstrate the ability to monitor the switches' performance in a per-packet way (in any pipeline we have programmed) and the fundamental workload-distributing ability based on the collected telemetry. The ultimate goal is to demonstrate how P4 devices and software-defined management can accelerate data exchange between end-users and Edge/Fog nodes by monitoring per-flow performance.



■ **Figure 8** Time that packets spent in the queue of switch 1 for traffic redirecting.

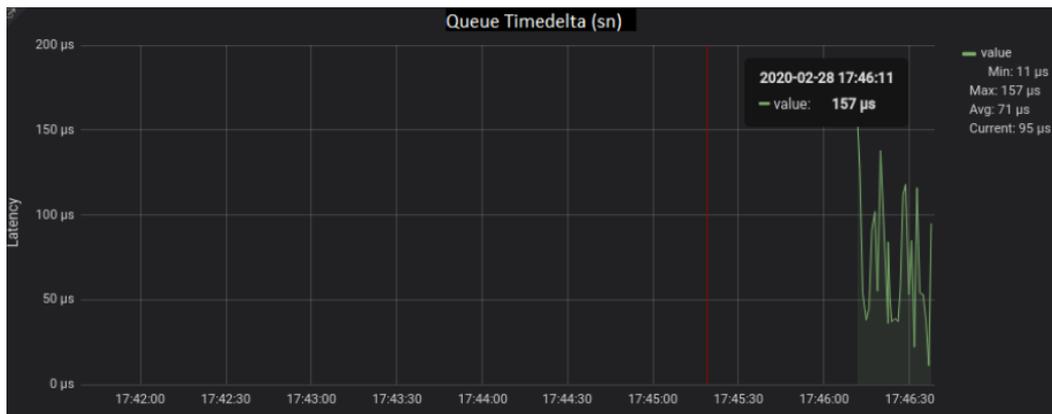
5 Conclusion

In this paper, we have shown how data telemetry detection using data plane programming works, and how can software-defined managing architectures fit into Fog computing. Although SDN technologies have not primarily been focused on Edge/Fog networks, we have seen more and more researches working in this area. The fact that the SDN data plane has evolved from the typical OpenFlow switches to P4 programmable ones creates a huge space for new protocols, data plane telemetry measurement, and network management. We have also demonstrated how INT-capable P4 switches work within networks, and how to collect telemetry data, demonstrating with simulations on how to collect per-packet queuing delay. Furthermore, a preliminary workload distribution based on the collected data telemetry illustrates how data plane programming can be used in network resource management. We propose the network managing architecture of this paper to utilize data telemetry for optimization of Fog network, because we strongly believe that the data-plane operation will conform to the distributing nature of Fog network. This also enables new traffic engineering ways of distributing computational resources at runtime. We believe that our work will lead to new methods that achieve a lower network latency, by altering traffic forwarding



■ **Figure 9** Time that packets spent in the queue of switch 2 for traffic redirecting.

12:10 Detection of Fog Network Data Telemetry Using Data Plane Programming



■ **Figure 10** Time that packets spent in the queue of switch n for traffic redirecting.

paths based on real-time network state. Furthermore, this method can offer third party applications (Inter-SDN controller communication, Time-Sensitive Networks, etc.) to benefit from it.

References

- 1 Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014. doi:10.1145/2656877.2656890.
- 2 P4 Language Consortium. Behavioral model (bmv2). URL: <https://github.com/p4lang/behavioral-model> [cited 2020-01-21].
- 3 Amir Vahid Dastjerdi, Harshit Gupta, Rodrigo N Calheiros, Soumya K Ghosh, and Rajkumar Buyya. Fog computing: Principles, architectures, and applications. In *Internet of things*, pages 61–75. Elsevier, 2016. doi:10.1016/B978-0-12-805395-9.00004-6.
- 4 Krittin Intharawijitr, Katsuyoshi Iida, and Hiroyuki Koga. Analysis of fog model considering computing and communication latency in 5g cellular networks. In *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pages 1–4. IEEE, 2016. doi:10.1109/PERCOMW.2016.7457059.
- 5 Vimalkumar Jeyakumar, Mohammad Alizadeh, Yilong Geng, Changhoon Kim, and David Mazières. Millions of little minions: Using packets for low latency network programming and visibility. *ACM SIGCOMM Computer Communication Review*, 44(4):3–14, 2014. doi:10.1145/2740070.2626292.
- 6 Rutvij H Jhaveri, Rui Tan, Arvind Easwaran, and Sagar V Ramani. Managing industrial communication delays with software-defined networking. In *2019 IEEE 25th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–11. IEEE, 2019. doi:10.1109/RTCSA.2019.8864557.
- 7 Changhoon Kim, Anirudh Sivaraman, Naga Katta, Antonin Bas, Advait Dixit, and Lawrence J Wobker. In-band network telemetry via programmable dataplanes. In *ACM SIGCOMM*, 2015.
- 8 Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 1–6, 2010. doi:10.1145/1868447.1868466.
- 9 Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008. doi:10.1145/1355734.1355746.

- 10 Mithun Mukherjee, Suman Kumar, Mohammad Shojafar, Qi Zhang, and Constandinos X Mavromoustakis. Joint task offloading and resource allocation for delay-sensitive fog networks. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2019. doi:10.1109/ICC.2019.8761239.
- 11 Subhadeep Sarkar and Sudip Misra. Theoretical modelling of fog computing: a green computing paradigm to support iot applications. *Iet Networks*, 5(2):23–29, 2016. doi:10.1049/iet-net.2015.0034.
- 12 William Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison-Wesley Longman Publishing Co., Inc., 1998.
- 13 Deze Zeng, Lin Gu, Song Guo, Zixue Cheng, and Shui Yu. Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system. *IEEE Transactions on Computers*, 65(12):3702–3712, 2016. doi:10.1109/TC.2016.2536019.

